
Lecture 5 – Authenticated encryption

TEK4500

25.09.2024

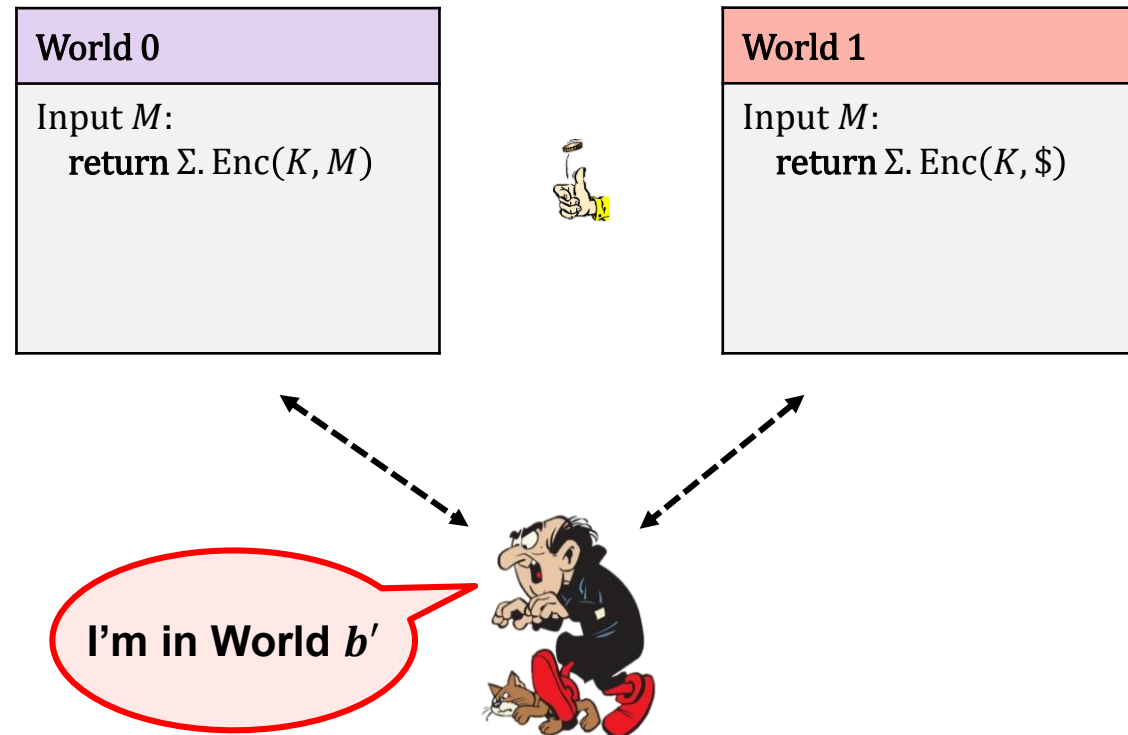
Håkon Jacobsen

hakon.jacobsen@its.uio.no

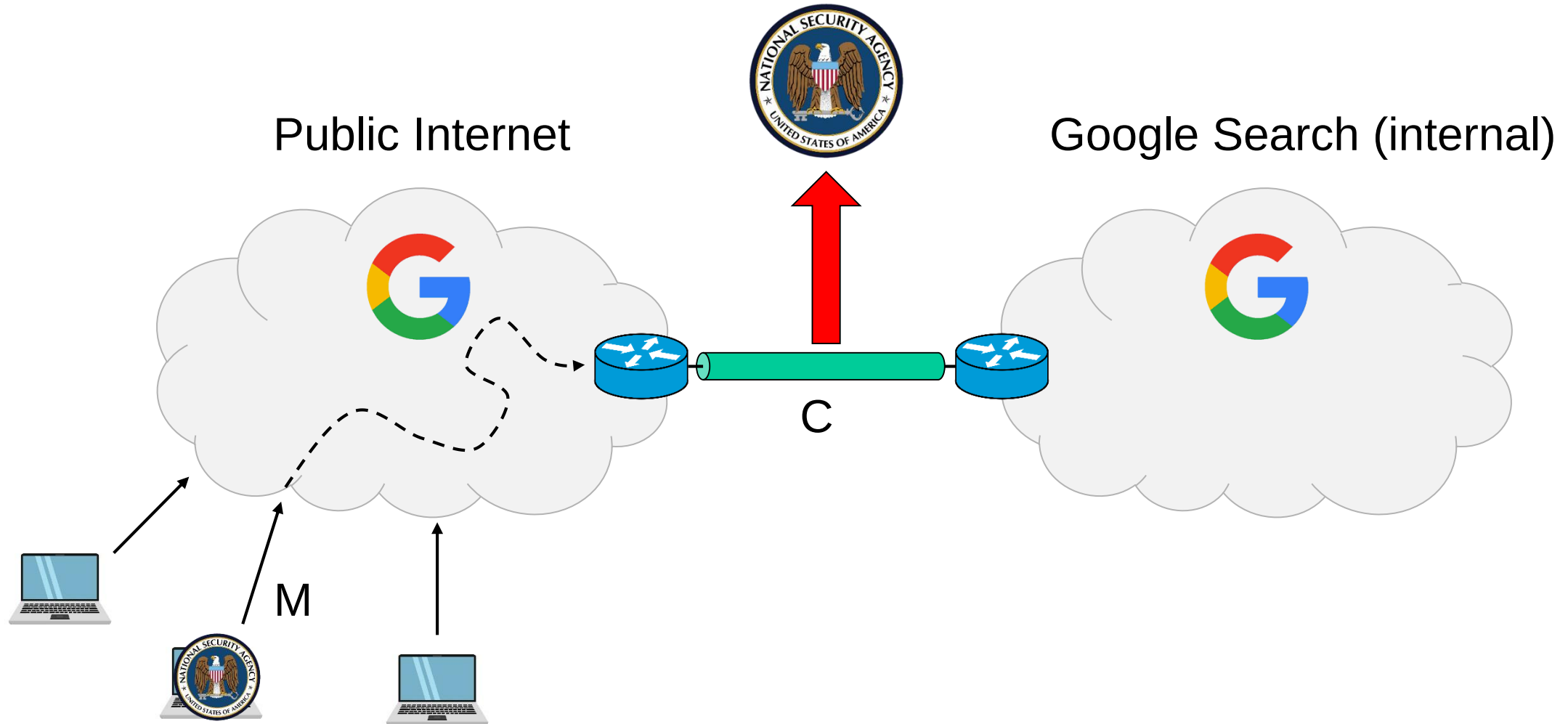
Basic goals of cryptography

	Message privacy	Message integrity / authentication
Symmetric keys	Symmetric encryption	Message authentication codes (MAC)
Asymmetric keys	Asymmetric encryption (a.k.a. public-key encryption)	Digital signatures

IND-CPA – Indistinguishability against chosen-plaintext attacks



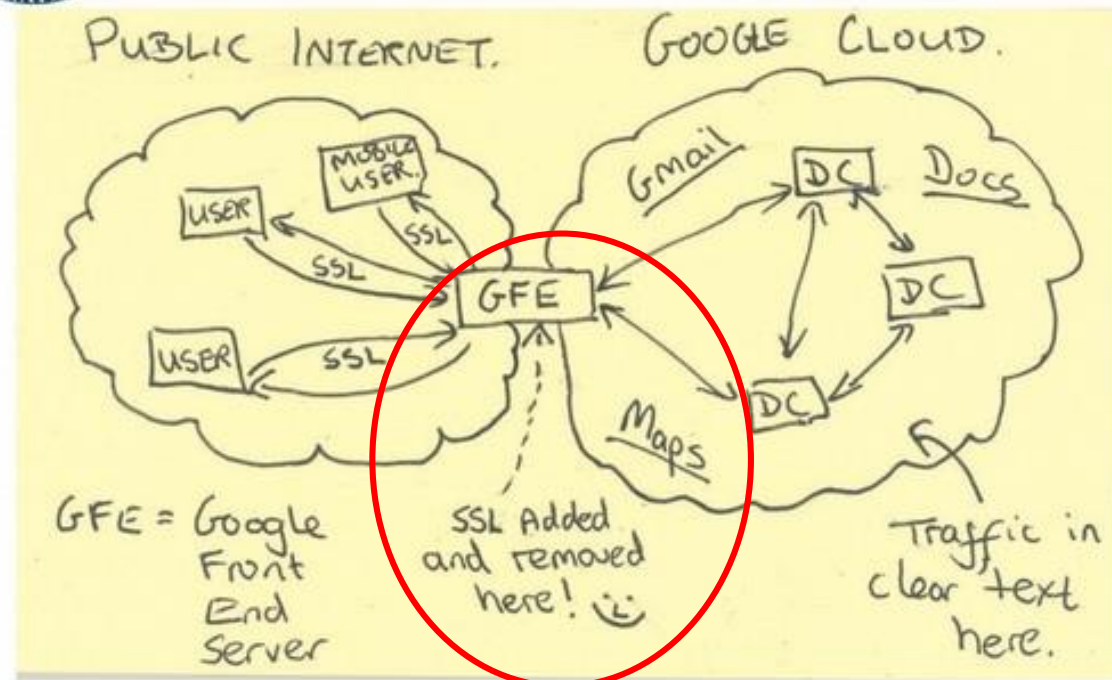
Chosen-plaintext attacks



Chosen-plaintext attacks

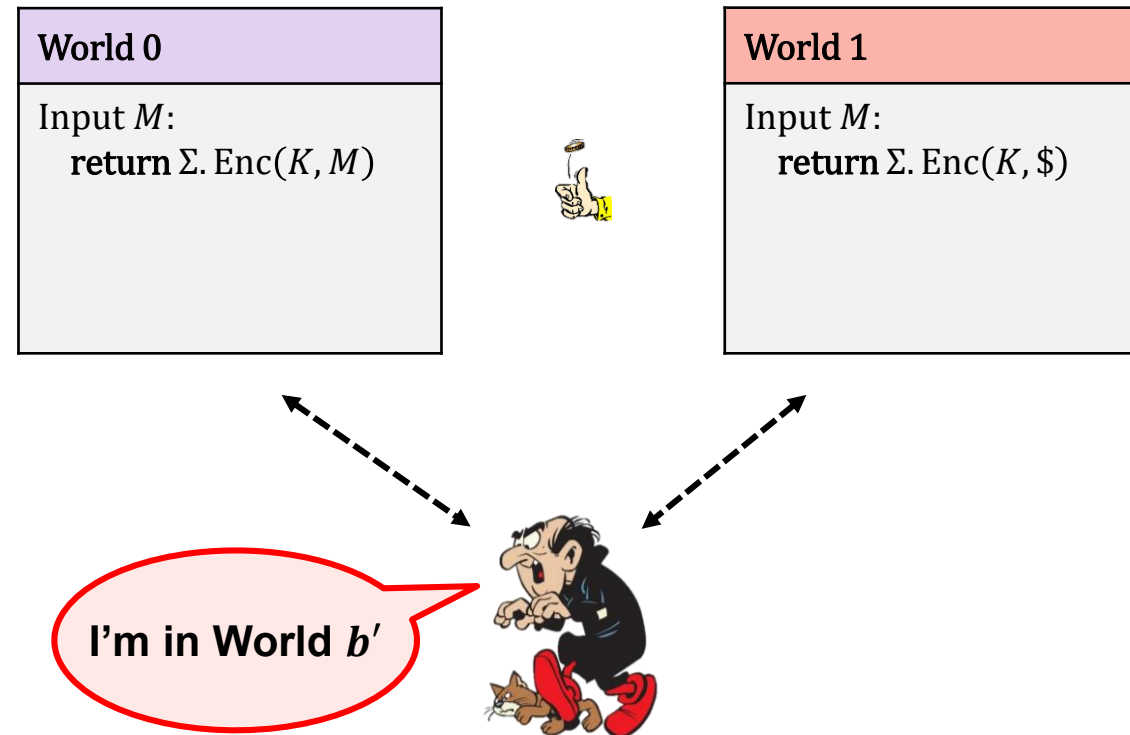


Current Efforts - Google

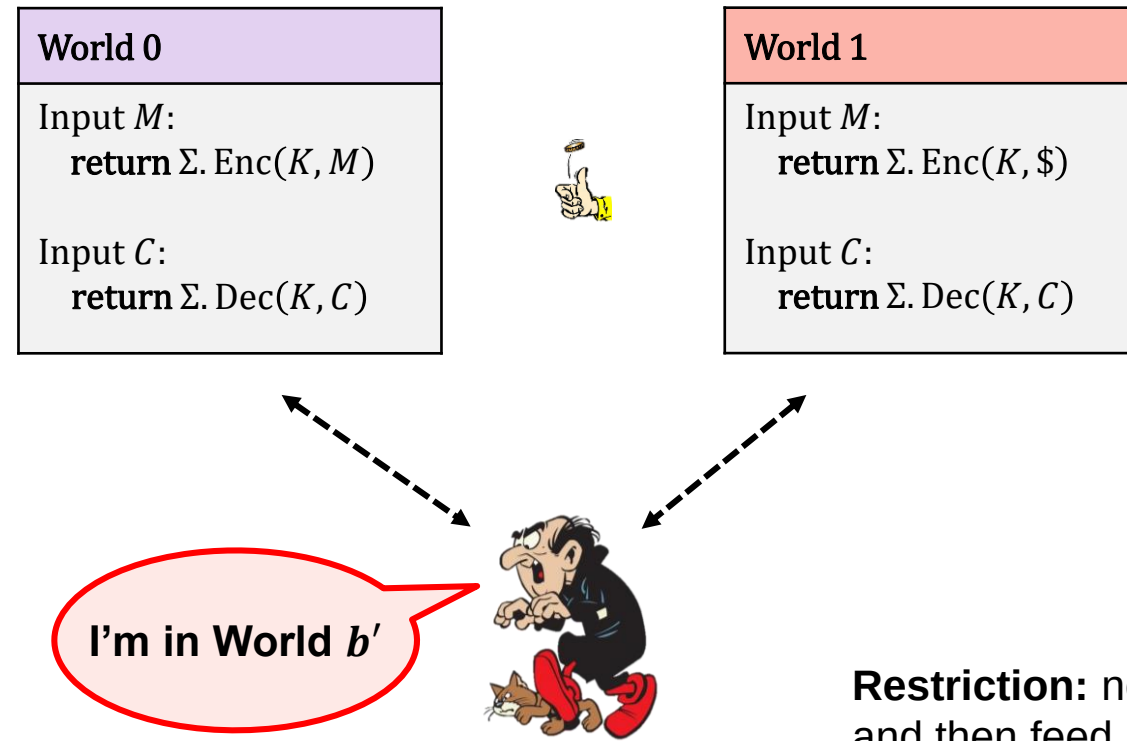


TOP SECRET//SI//NOFORN

IND-CPA – Indistinguishability against chosen-plaintext attacks



IND-CCA – Indistinguishability against chosen-ciphertext attacks



Xilinx FPGA – Starbleed attack

[Security Tools and Resources](#) [Privacy Policy](#) [FAQ's](#)

ROOTDAEMON.COM

SECURITY NEWS

Unpatchable 'Starbleed' Attack Exposes Critical Design Flaw

BY ROOTDAEMON © APRIL 21, 2020



A newly discovered unpatchable hardware flaw could allow an attacker to break bitstream encryption on Xilinx 7-Series FPGAs.

The Unpatchable Silicon: A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs

Maik Ender*, Amir Moradi† and Christof Paar*‡

*Horst Goertz Institute for IT Security, Ruhr University Bochum, Germany
†Max Planck Institute for Cyber Security and Privacy, Germany

Abstract

The security of FPGAs is a crucial topic, as any vulnerability within the hardware can have severe consequences, if they are used in a secure design. Since FPGA designs are encoded in a bitstream, securing the bitstream is of the utmost importance. Adversaries have many motivations to recover and manipulate the bitstream, including design cloning, IP theft, manipulation of the design, or design subversions e.g., through hardware Trojans. Given that FPGAs are often part of cyber-physical systems e.g., in aviation, medical, or industrial devices, this can even lead to physical harm. Consequently, vendors have introduced bitstream encryption, offering authenticity and confidentiality. Even though attacks against bitstream encryption have been proposed in the past, e.g., side-channel analysis and probing, these attacks require sophisticated equipment and considerable technical expertise.

In this paper, we introduce novel low-cost attacks against the Xilinx 7-Series (and Virtex-6) bitstream encryption, resulting in the total loss of authenticity and confidentiality. We exploit a design flaw which piecewise leaks the decrypted bitstream. In the attack, the FPGA is used as a decryption oracle, while only access to a configuration interface is needed. The attack does not require any sophisticated tools and, depending on the target system, can potentially be launched remotely. In addition to the attacks, we discuss several countermeasures.

1 Introduction

Nowadays, Field Programmable Gate Arrays (FPGAs) are common in consumer electronic devices, aerospace, financial computing, and military applications. Additionally, given the trend towards a connected world, data-driven practices, and artificial intelligence, FPGAs play a significant role as hardware platforms deployed in the cloud and in end devices. Hence, trust in the underlying platform for all these applications is vital. Altera, who are (together with Xilinx) the FPGA market leader, was acquired by Intel in 2015.

FPGAs are reprogrammable ICs, containing a repetitive logic area with a few hundred up to millions of reprogrammable gates. The bitstream configures this logic area; in analogy to software, the bitstream can be considered the 'binary code' of the FPGA. On SRAM-based FPGAs, which are the dominant type of FPGA in use today, the bitstream is stored on an external non-volatile memory and loaded into the FPGA during power-up.

In order to protect the bitstream against malicious actors, its confidentiality and authenticity must be assured. If an attacker has access to the bitstream and breaks its confidentiality, he can reverse-engineer the design, clone intellectual property, or gather information for subsequent attacks e.g., by finding cryptographic keys or other design aspects of a system. If the adversary succeeds in violating the bitstream authenticity, he can then change the functionality, implant hardware Trojans, or even physically destroy the system in which the FPGA is embedded by using configuration outside the specifications. These problems are particularly relevant since access to bitstream is often effortlessly possible due to the fact that, for the vast majority of devices, it resides in the external non-volatile memory, e.g., flash chips. This memory can often either be read out directly, or the adversary wiretaps the FPGA's configuration bus during power-up. Alternatively, a microcontroller can be used to configure the FPGA, and consequently, the microcontroller's firmware includes the bitstream. When the adversary gains access to the microcontroller, he also gains access to the configuration interface and the bitstream. Thus, if the microcontroller is connected to a network, remotely attacking the FPGA becomes possible.

In order to protect the design, the major FPGA vendors introduced bitstream encryption around the turn of the millennium, a technique which nowadays is available in most mainstream devices [1,56]. In this paper, we investigate the security of the Xilinx 7-Series and Virtex-6 bitstream encryption. On these devices, the bitstream encryption provides authenticity by using an SHA-256 based HMAC and also provides confidentiality by using CBC-AES-256 for encryption. By our attack, we can circumvent the bitstream encryption and decrypt an assumedly secure bitstream on all Xilinx 7-Series devices completely and on the Virtex-6 devices partially. Adversaries have many motivations to recover and manipulate the bitstream, including design cloning, IP theft, manipulation of the design, or design subversions e.g., through hardware Trojans. Given that FPGAs are often part of cyber-physical systems e.g., in aviation, medical, or industrial devices, this can even lead to physical harm. Consequently, vendors have introduced bitstream encryption, offering authenticity and confidentiality. Even though attacks against bitstream encryption have been proposed in the past, e.g., side-channel analysis and probing, these attacks require sophisticated equipment and considerable technical expertise.

USENIX Association

29th USENIX Security Symposium 1803

[HOME](#) [NEWS](#) [HACKING TOOLS](#) [CTF](#) [SHOP](#) [COURSES](#) [AFFILIATES](#)



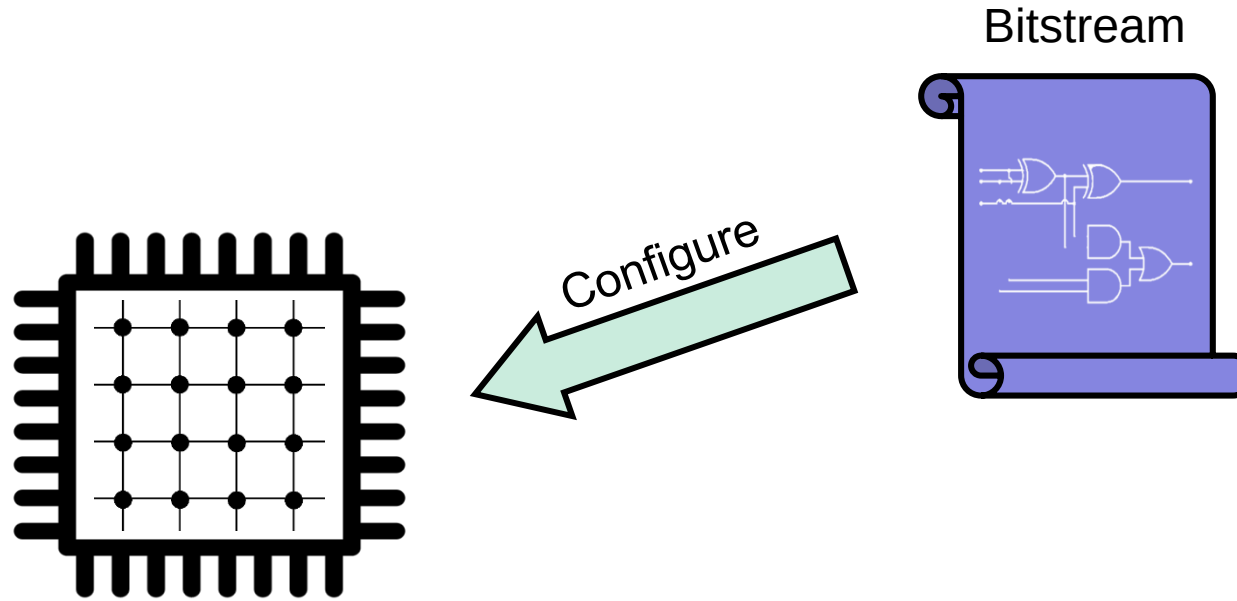
News News Vulnerabilities

Vulnerability Discovered In FPGA

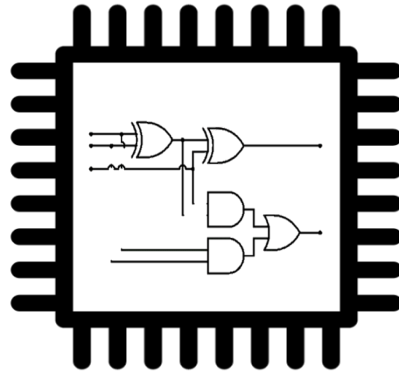
bitstream, bitstream encryption, bug, Chips, decryption, decryption key, Gate Arrays, flaw, FPGA chips, Hardware, hardware encryption, hardware

vulnerability, Starbleed, Starbleed vulnerability, vulnerability

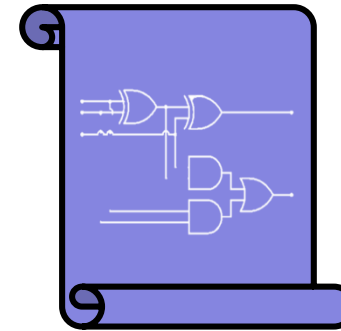
FPGA – Field Programmable Gate Array



FPGA – Field Programmable Gate Array

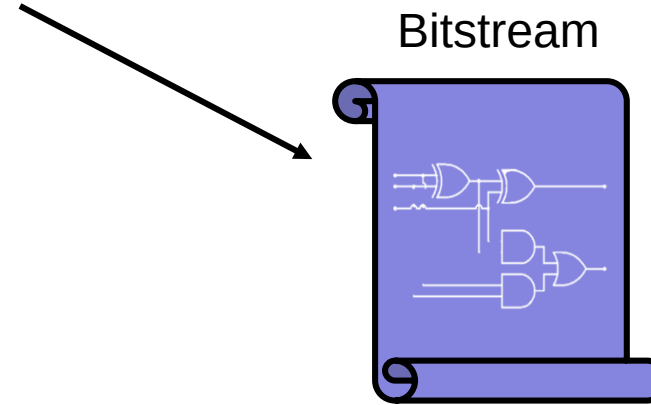
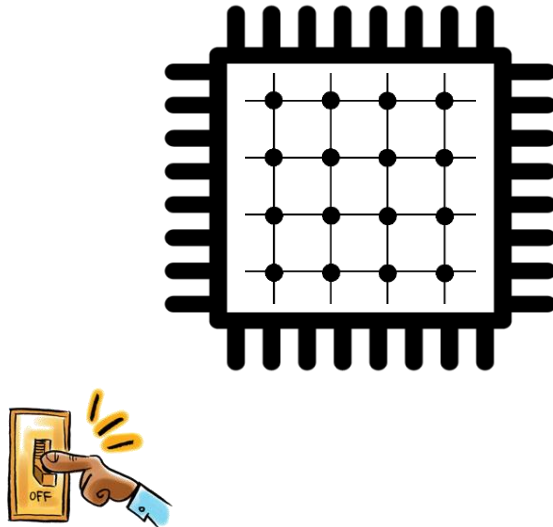


Bitstream



FPGA – Field Programmable Gate Array

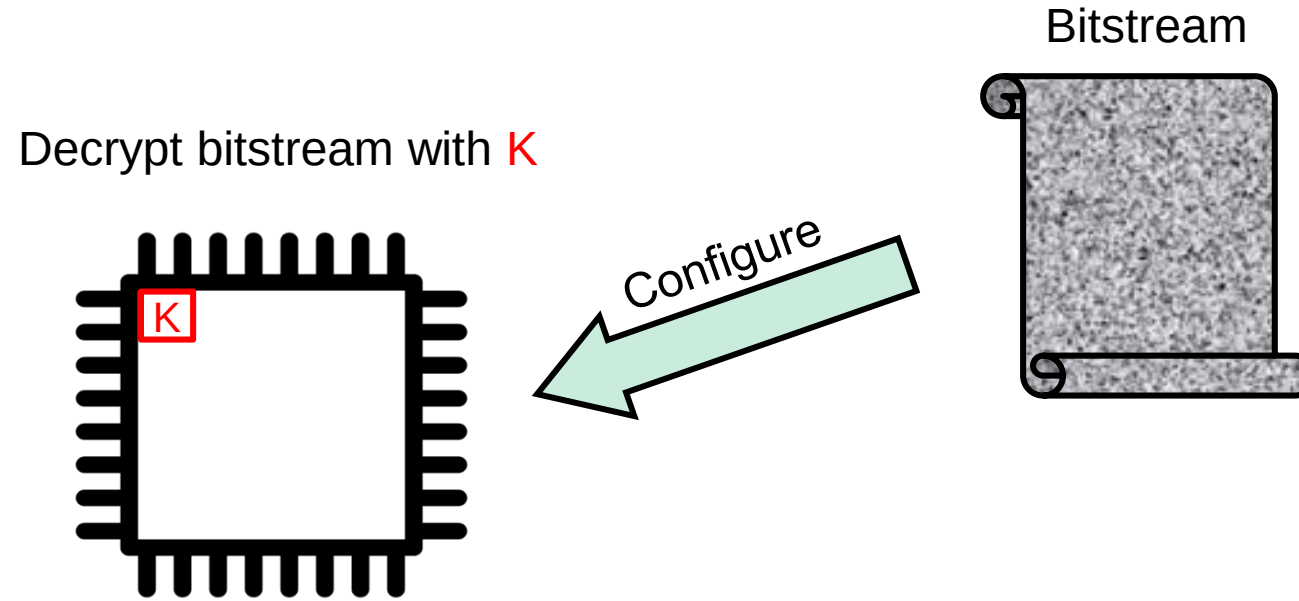
Bitstream design typically a business secret
(or even a national/military secret)



FPGA applications:

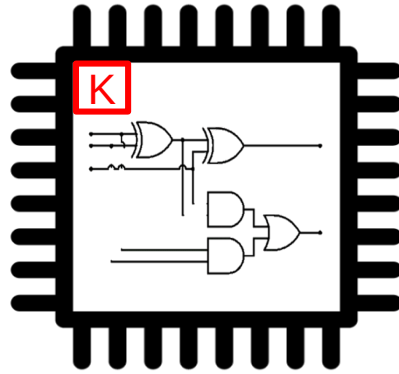
- Aerospace and avionics
- Digital signal processors
- Defense and military
- Medical devices
- General hardware accelerators (e.g. cryptography)

FPGA – Field Programmable Gate Array

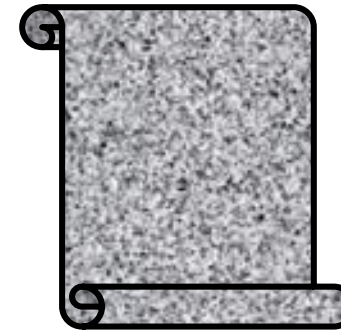


FPGA – Field Programmable Gate Array

Decrypt bitstream with **K**

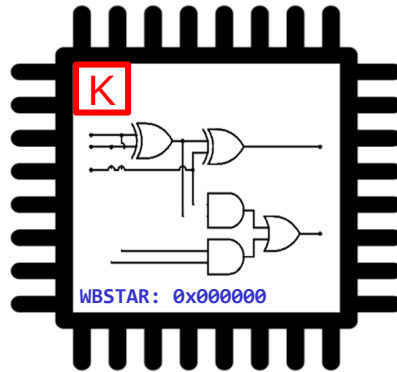


Bitstream



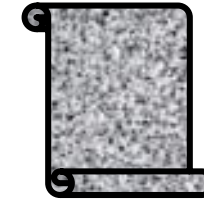
Xilinx Starbleed attack

Decrypt bitstream with **K**



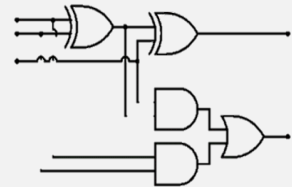
WBSTAR = **W**arm-**B**oot **S**tart-address

Bitstream

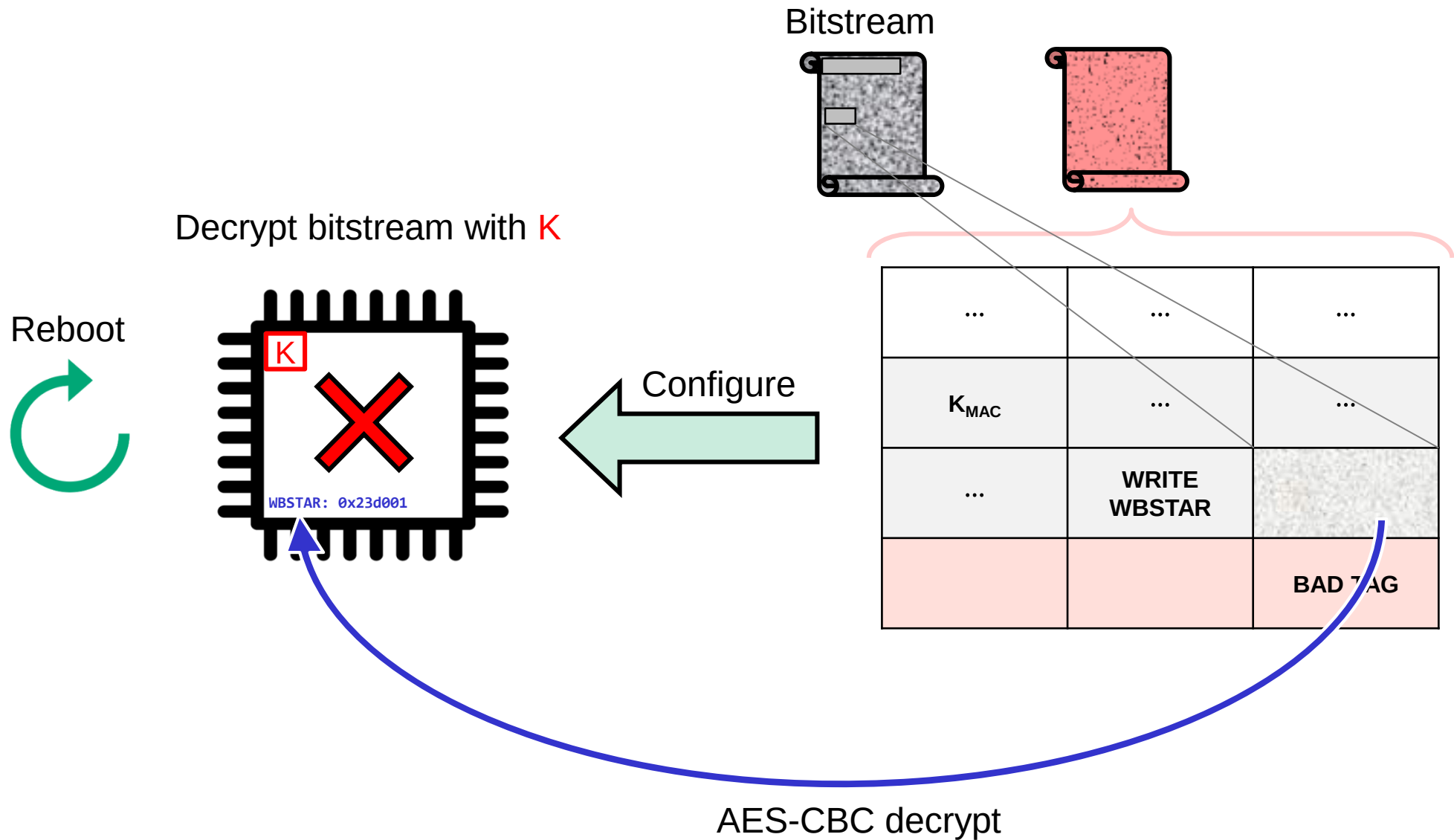


Header

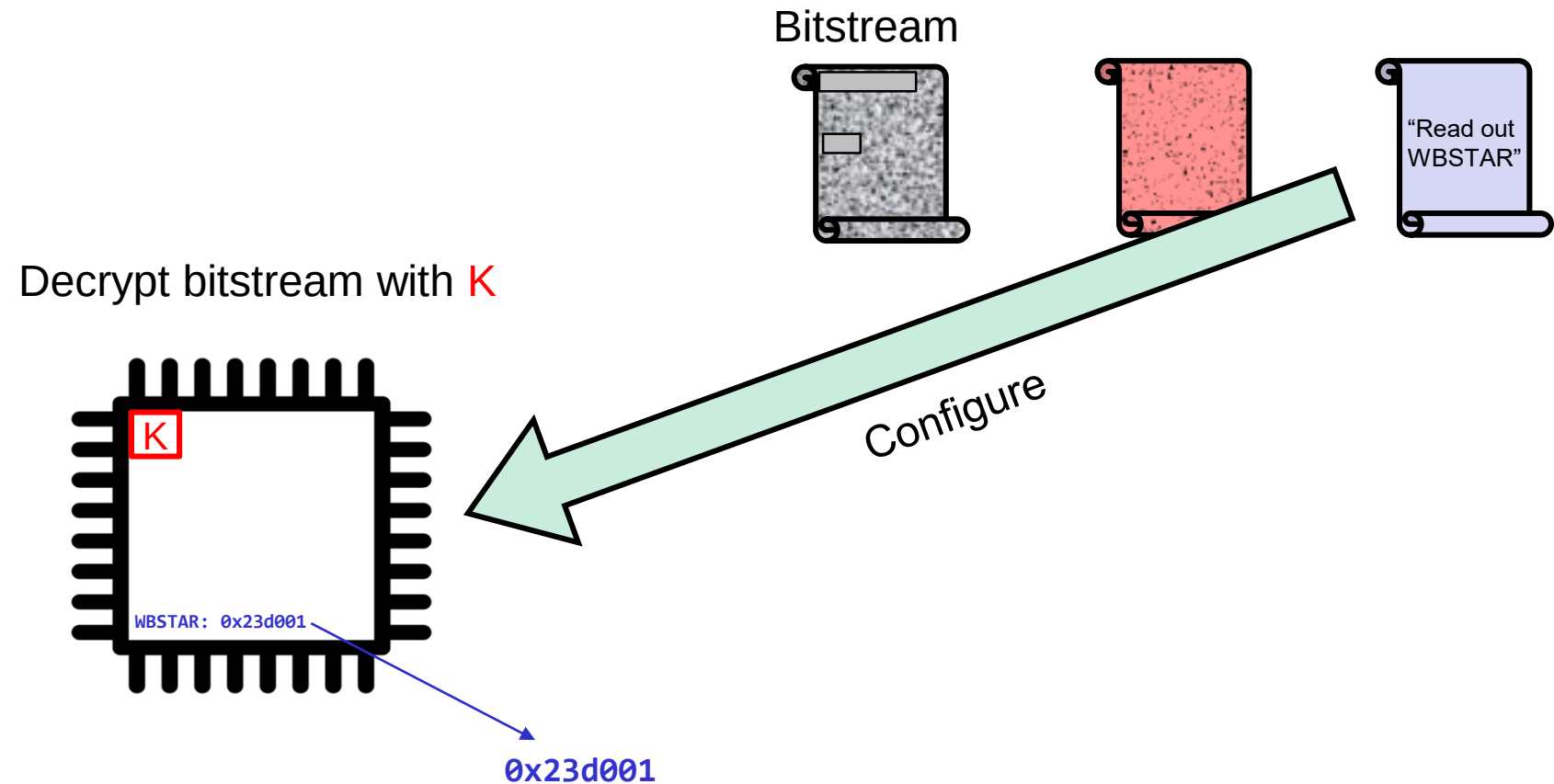
AES-CBC
encrypted

...	...	...
K _{MAC}	...	...
...	WRITE WBSTAR	0x00000000
		
		MAC TAG

Xilinx Starbleed attack

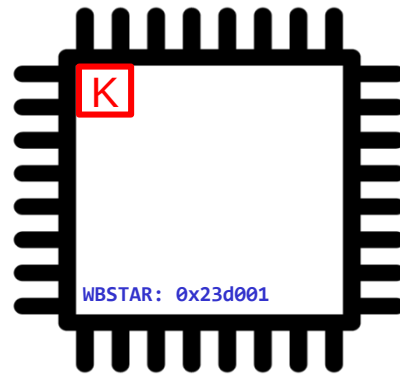


Xilinx Starbleed attack

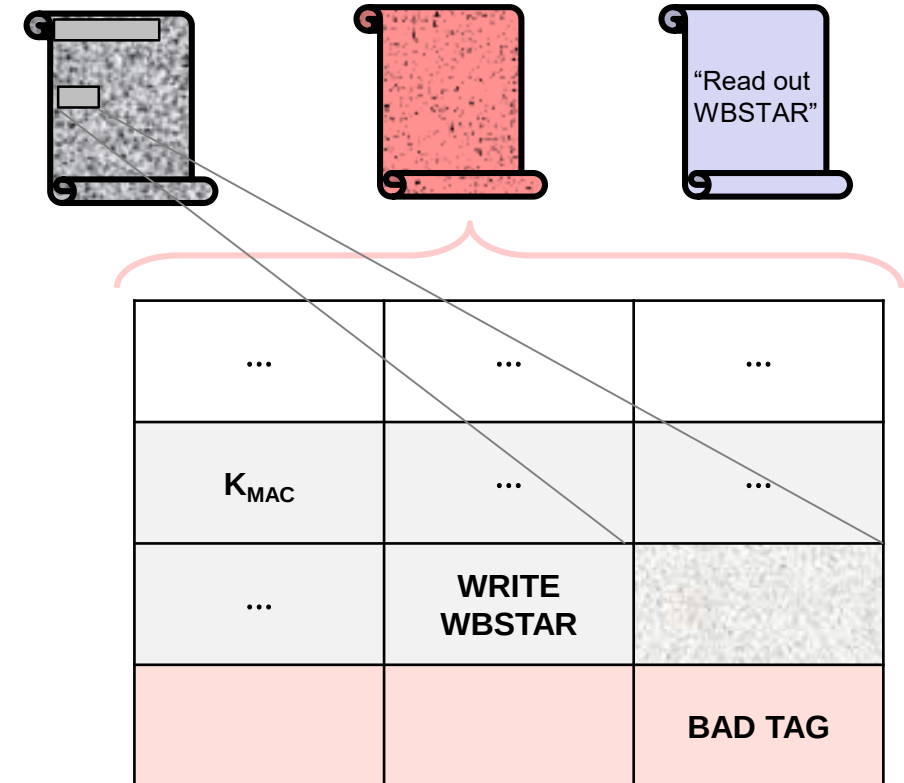


Xilinx Starbleed attack

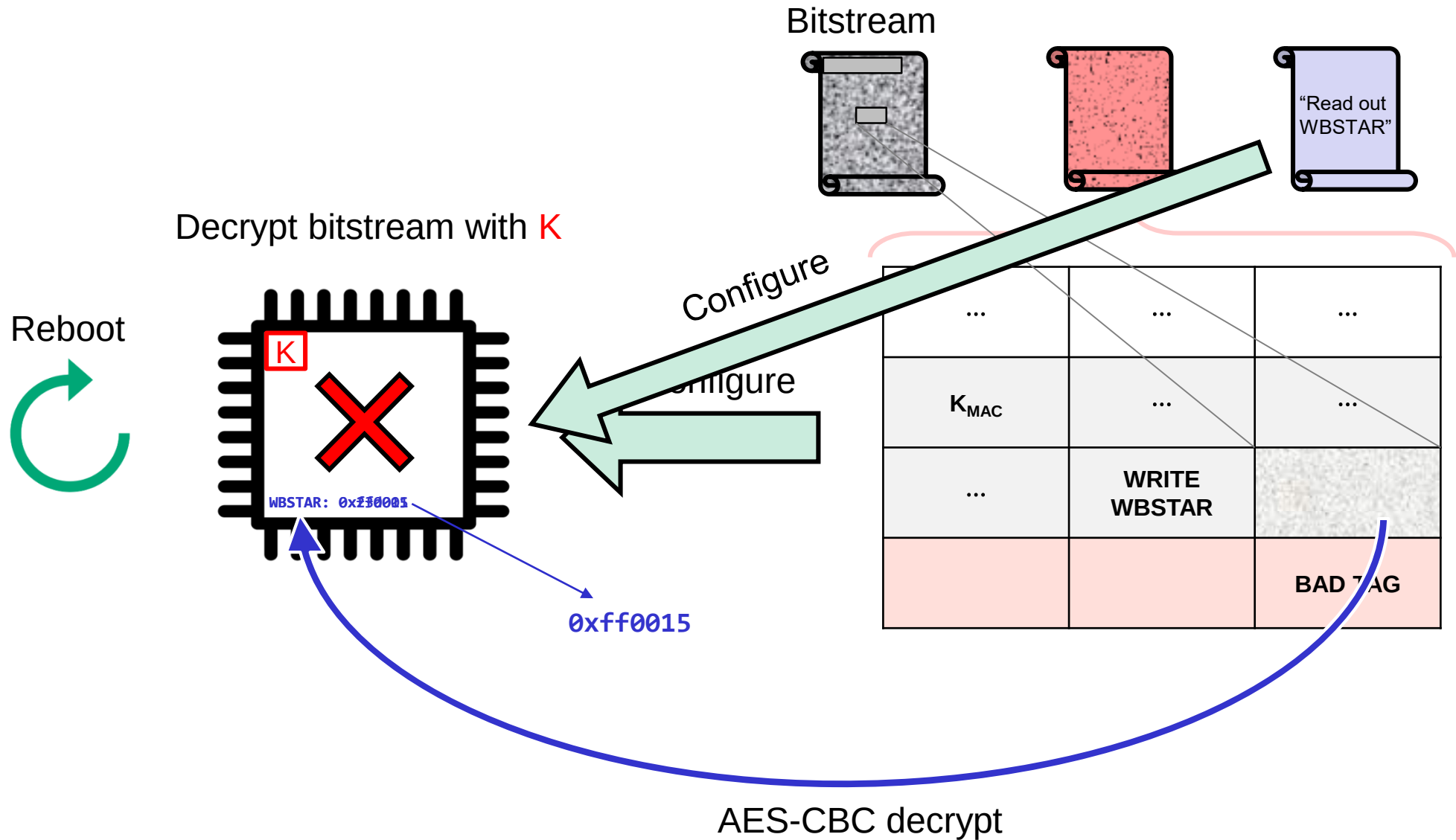
Decrypt bitstream with **K**



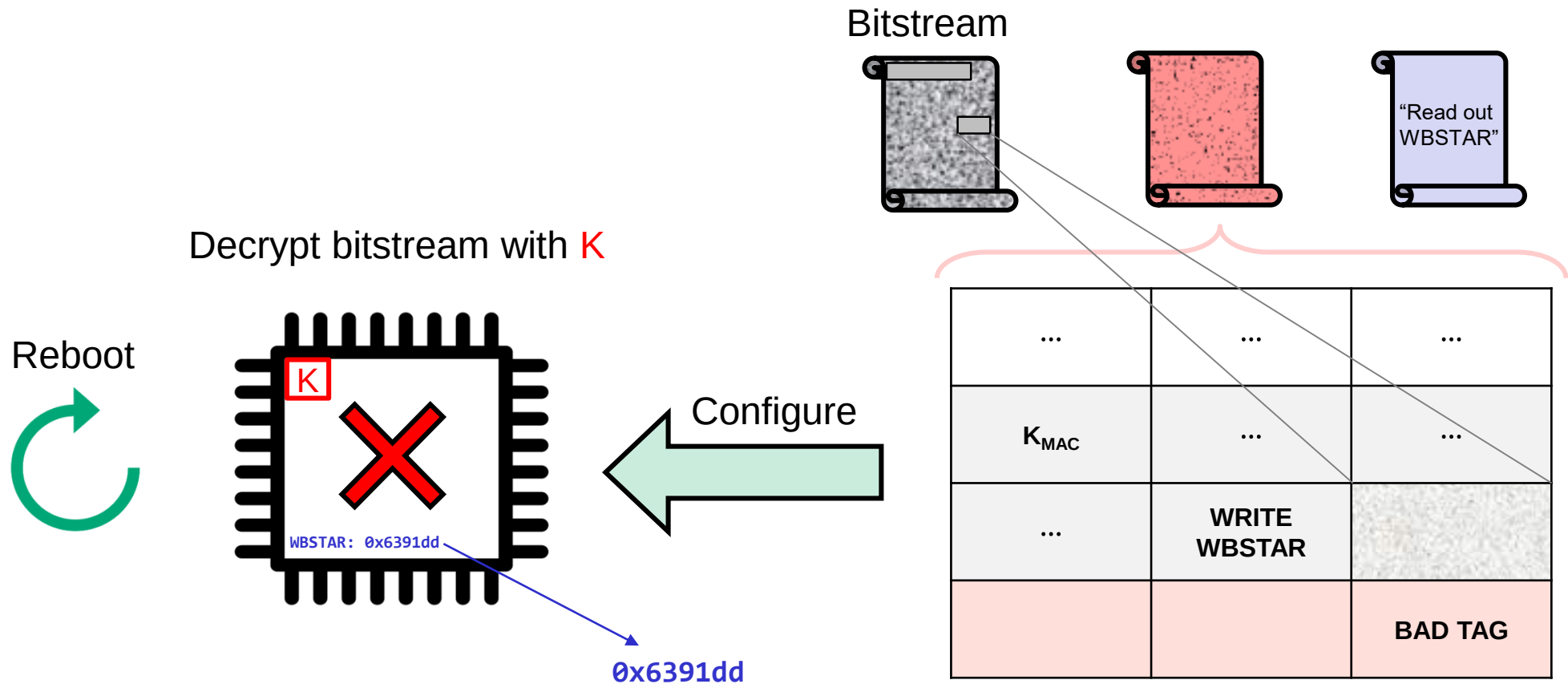
Bitstream



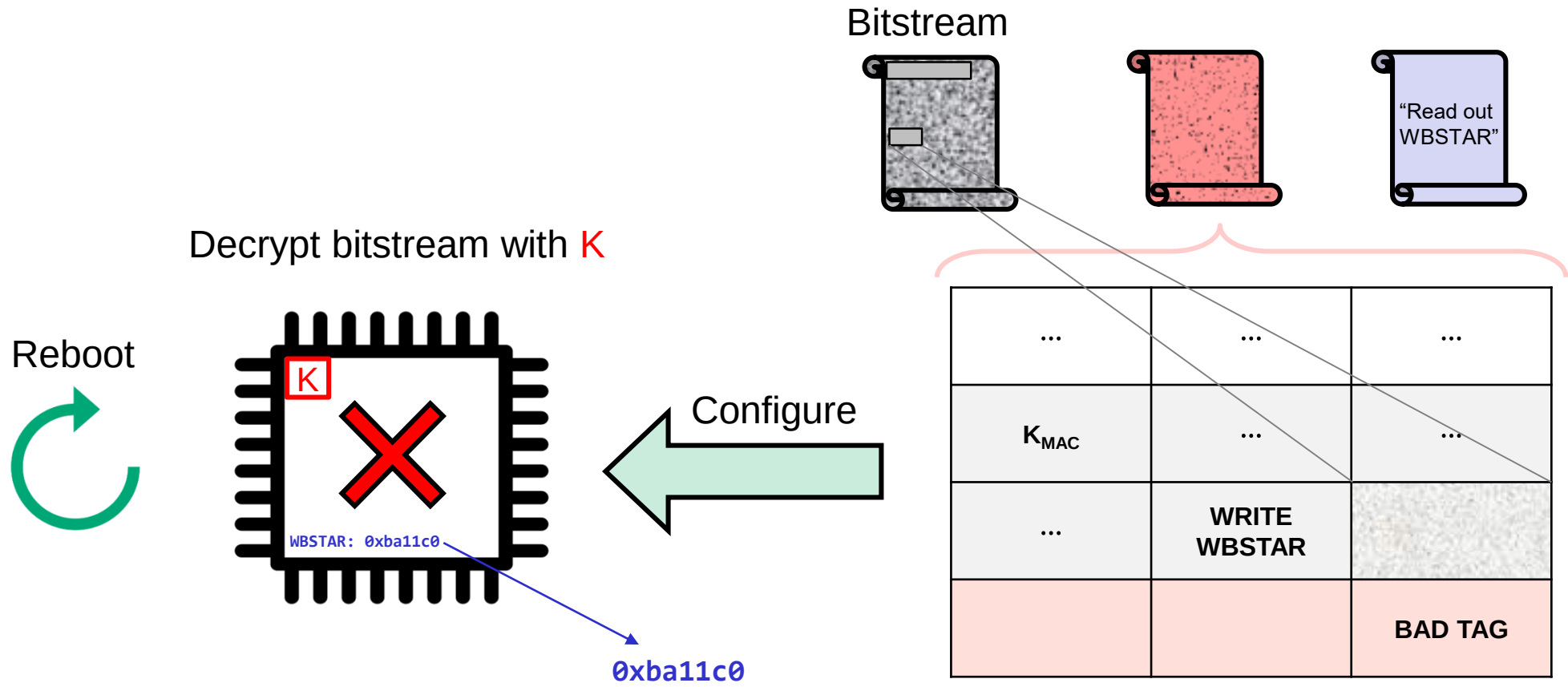
Xilinx Starbleed attack



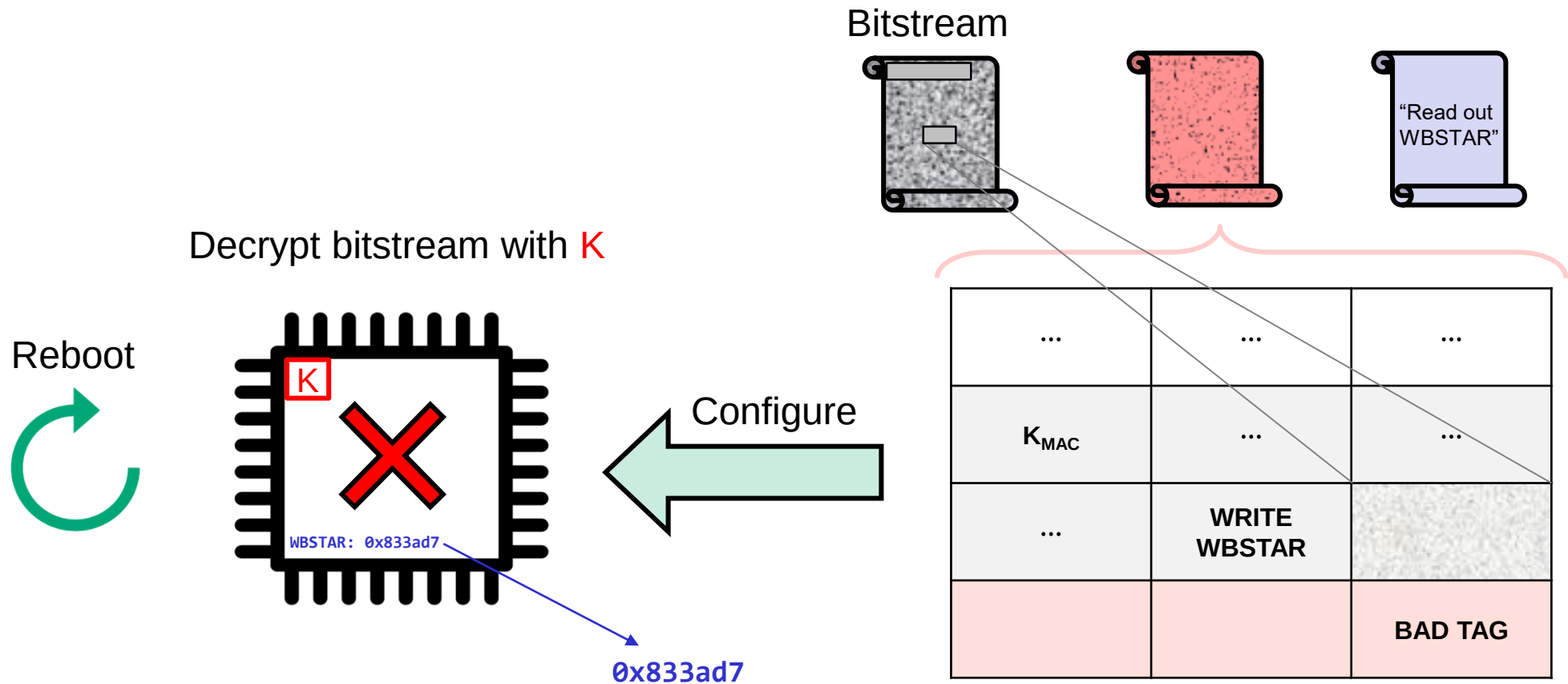
Xilinx Starbleed attack



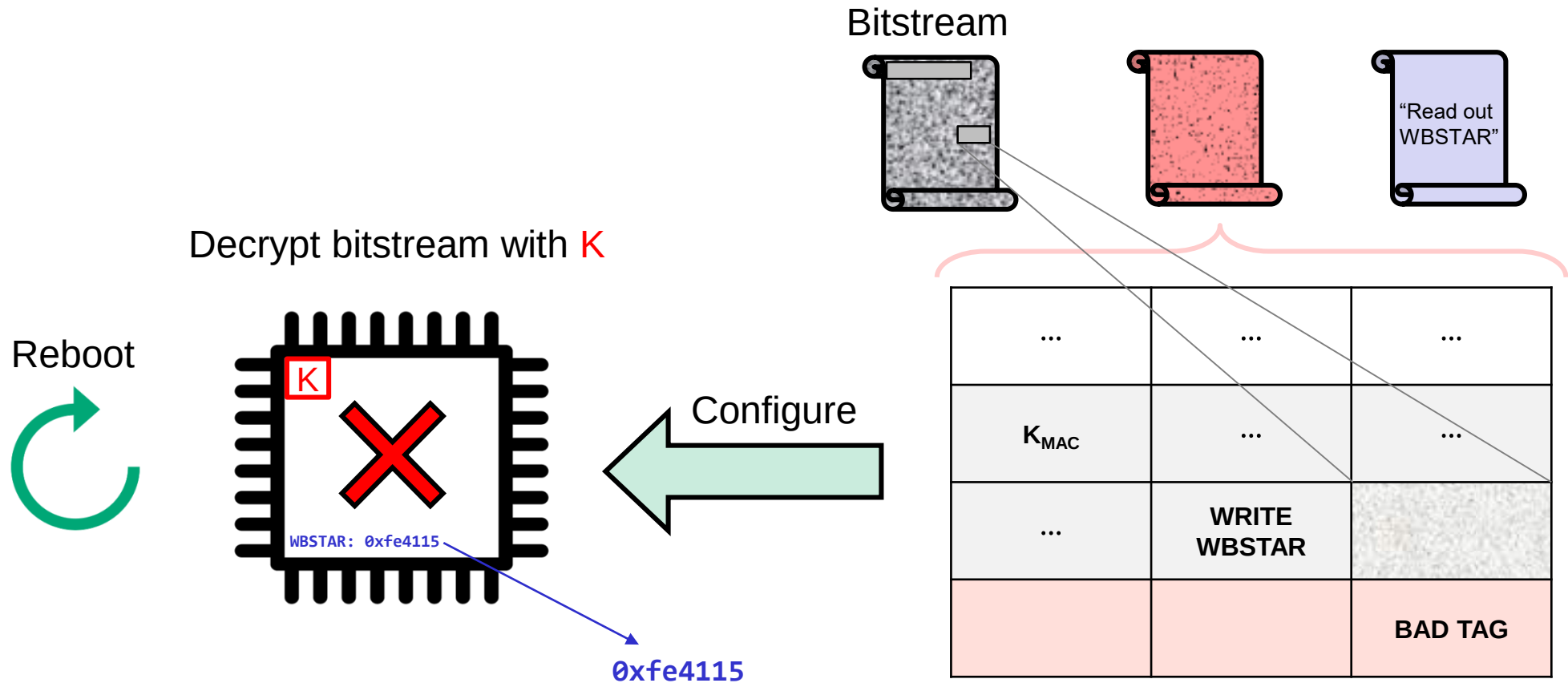
Xilinx Starbleed attack



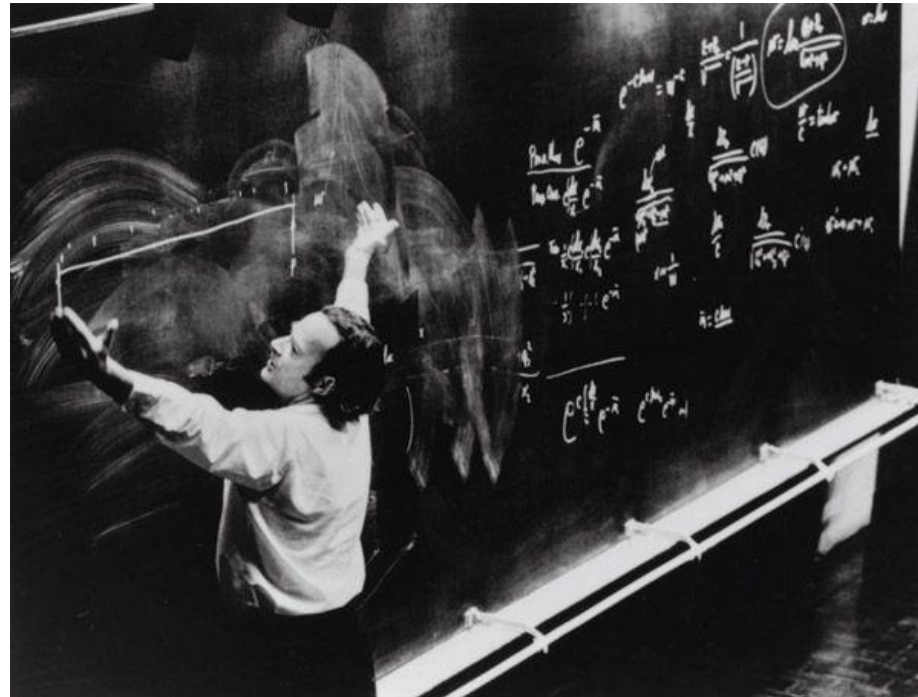
Xilinx Starbleed attack



Xilinx Starbleed attack



Time to fully decrypt bitstream: 26 hours



Authenticated encryption

Authenticated encryption

- **Authenticated encryption:** privacy *and* integrity from a single primitive:
- Syntactically (almost) the same as a normal encryption scheme

Authenticated encryption – syntax

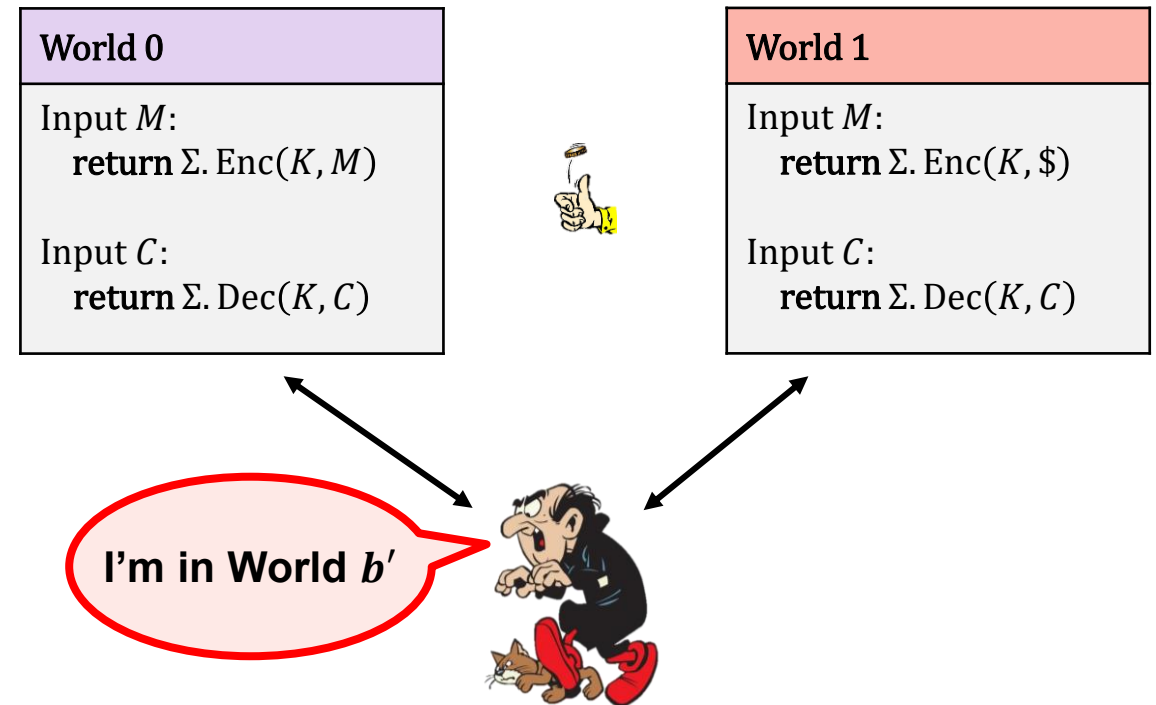
$$\text{Enc} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, M) = \text{Enc}_K(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, C) = \text{Dec}_K(C) = M / \perp$$

Authenticated encryption – security



Restriction: not allowed to query $\text{Enc}(K, \cdot)$ and then feed result to $\text{Dec}(K, \cdot)$ unmodified

Authenticated encryption – security

$\text{Exp}_{\Sigma}^{\text{ae}}(A)$

1. $b \xleftarrow{\$} \{0,1\}$
2. Ciphertexts $\leftarrow []$
3. $K \xleftarrow{\$} \Sigma.\text{KeyGen}$
4. $b' \leftarrow A^{\mathcal{E}(\cdot), \mathcal{D}(\cdot)}$
5. **return** $b' \stackrel{?}{=} b$

$\mathcal{E}(M)$

1. $R \xleftarrow{\$} \{0,1\}^{|M|}$
2. $C_0 \leftarrow \Sigma.\text{Enc}(K, M)$
3. $C_1 \leftarrow \Sigma.\text{Enc}(K, R)$
4. Ciphertexts.add(C_b)
5. **return** C_b

$\mathcal{D}(C)$

1. **if** $C \in \text{Ciphertexts}$ **then** // cheating!
2. **return** $\perp$
3. $M_0 \leftarrow \Sigma.\text{Dec}(K, C)$
4. $M_1 \leftarrow \perp$
5. **return** M_b

World 0

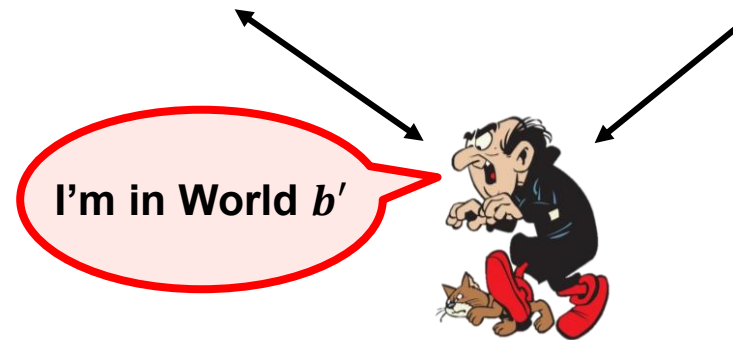
Input M :
return $\Sigma.\text{Enc}(K, M)$

Input C :
return $\Sigma.\text{Dec}(K, C)$

World 1

Input M :
return $\Sigma.\text{Enc}(K, \$)$

Input C :
return $\perp$



Restriction: not allowed to query $\text{Enc}(K, \cdot)$ and then feed result to $\text{Dec}(K, \cdot)$ unmodified

Definition: The **AE-advantage** of an adversary A is

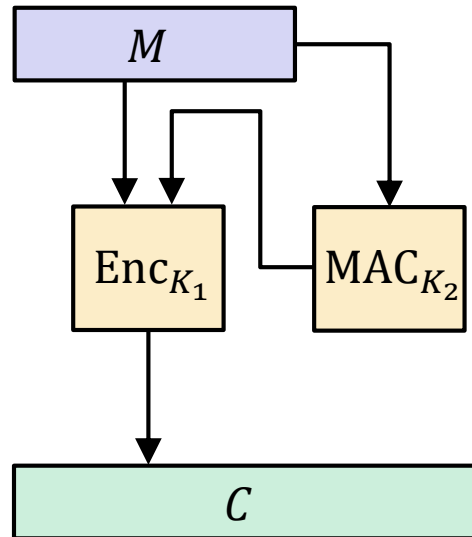
$$\text{Adv}_{\Sigma}^{\text{ae}}(A) = 2 \cdot |\Pr[\text{Exp}_{\Sigma}^{\text{ae}}(A) \Rightarrow \text{true}] - 1/2|$$

AE security definition – implications

- **Privacy:** adversary cannot distinguish encryption of real messages from encryption random messages
- **Integrity:** adversary is not able to **forge** ciphertexts: any ciphertext *not* produced by the legitimate sender will decrypt to $\perp$
- Does *not* protect against **replay attacks**

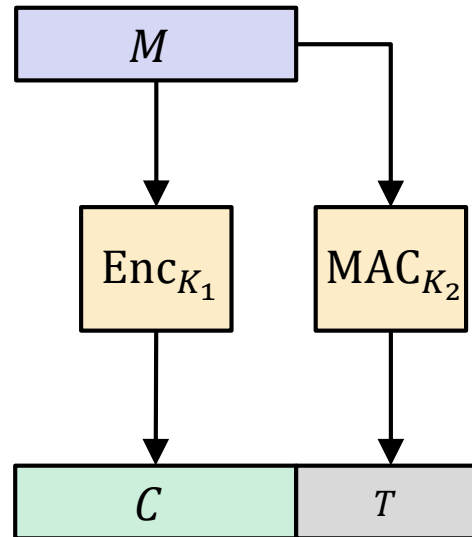
Generic composition: AE from Encryption + MAC

MAC-then-Encrypt (MtE)



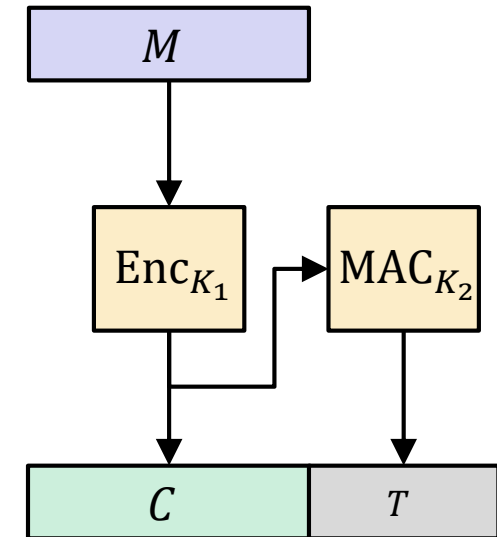
Used in TLS 1.2

Encrypt-and-MAC (E&M)



Used in SSH

Encrypt-then-MAC (EtM)



Used in IPsec

Generic composition: AE secure?

- IND-CPA secure UF-CMA secure

↓ ↓

E&M: $C || T \leftarrow \text{Enc}(K_1, M) || \text{MAC}(K_2, M)$

⏟

AE secure?

✗

$$\text{Enc}(K_1, M) || \overbrace{M || T}^{\text{MAC}(K_2, M)}$$

- MtE: $C \leftarrow \text{Enc}(K_1, M || \text{MAC}(K_2, M))$

✗

$$\underbrace{0000 || C}_{\text{Enc}(K_1, M || \dots)}$$

- EtM: $C || T \leftarrow \text{Enc}(K_1, M) || \text{MAC}(K_2, C)$

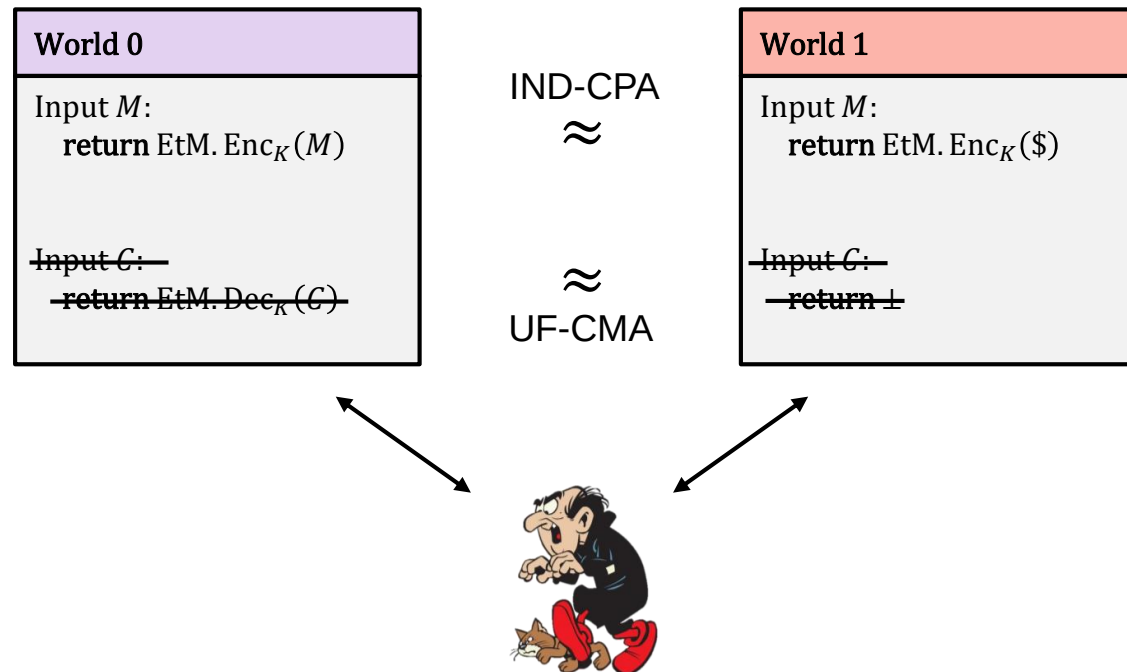
✓

EtM – security

Theorem: for any AE adversary A against EtM there are adversaries B and C against Enc and MAC such that

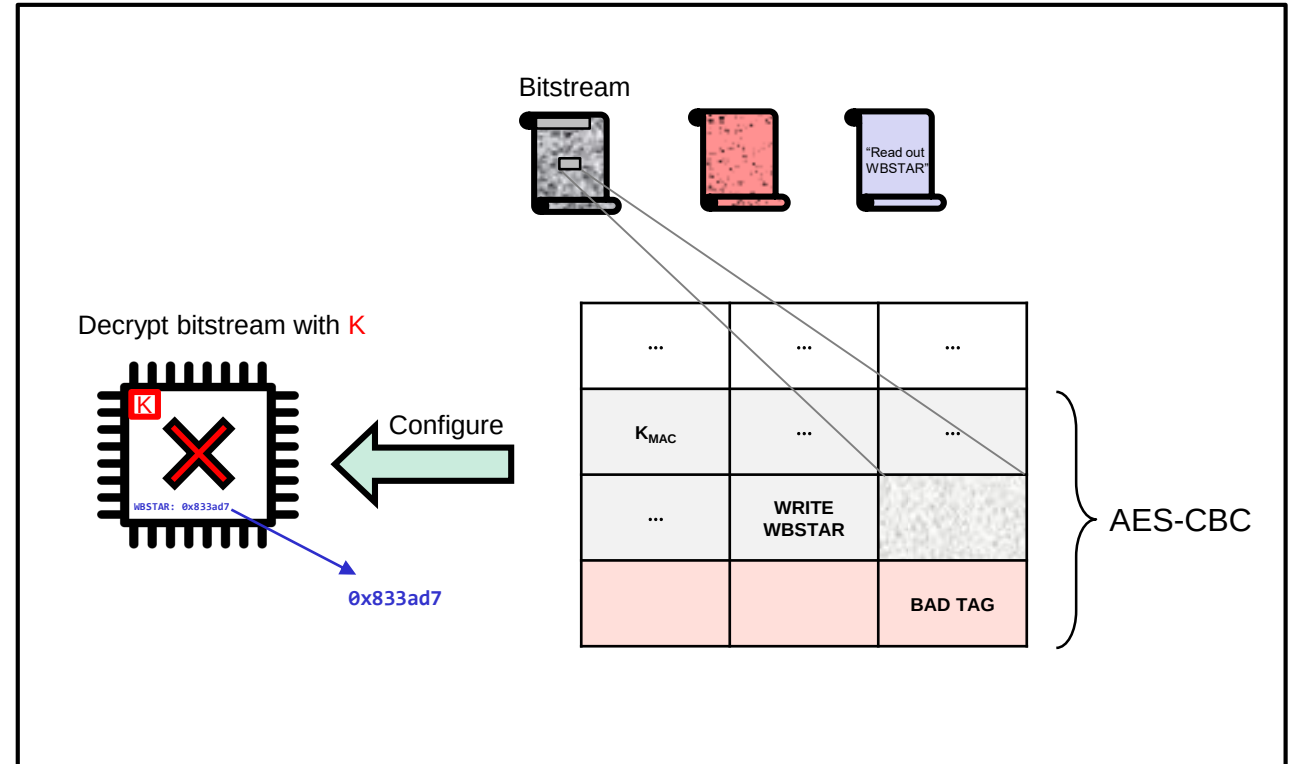
$$\mathbf{Adv}_{\text{EtM}}^{\text{ae}}(A) \leq \mathbf{Adv}_{\text{Enc}}^{\text{ind-cpa}}(B) + 2 \cdot \mathbf{Adv}_{\text{MAC}}^{\text{uf-cma}}(C)$$

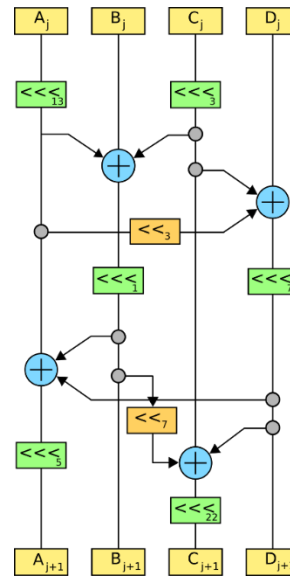
Proof sketch:



CCA-attacks – revisited

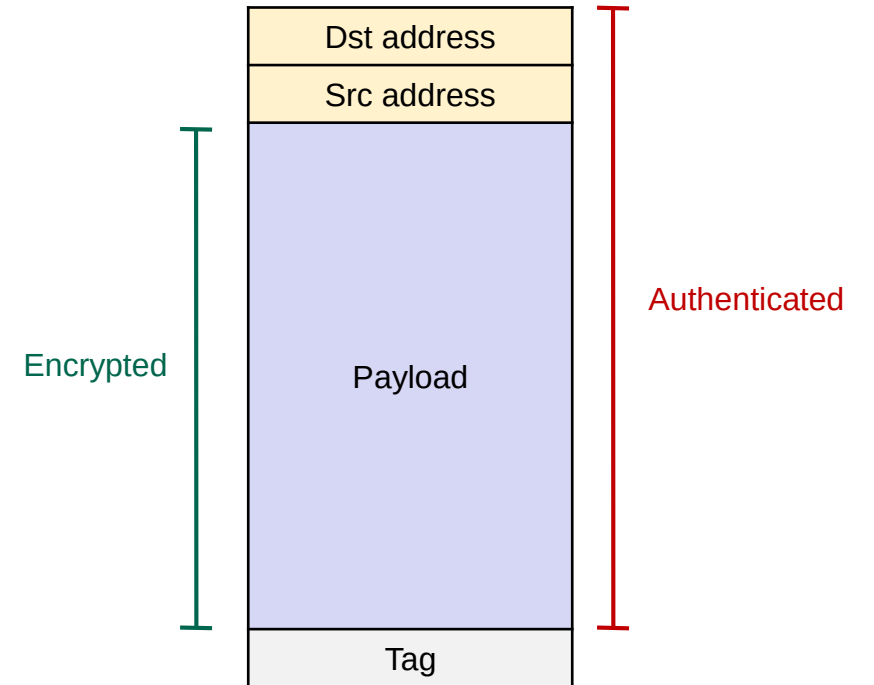
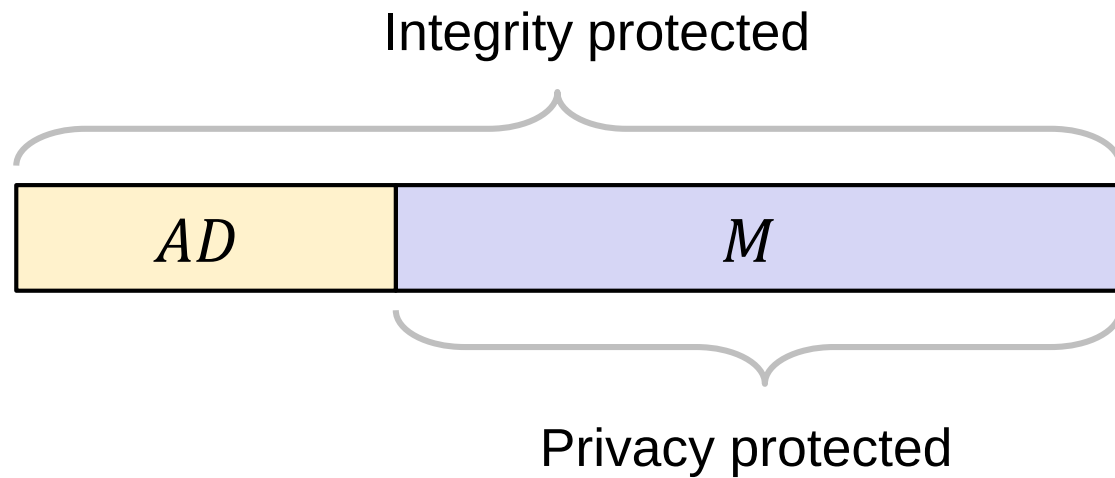
- MtE used in Starbleed attack
- Gave rise to (partial) decryption oracle
- Attack would not have been possible with an AE secure scheme





AEAD – Authenticated Encryption with Associated Data

AEAD – AE with associated data (AD)



- AD – data that can't be encrypted but still need integrity protection
 - Headers in protocols
 - Configuration data
 - Metadata

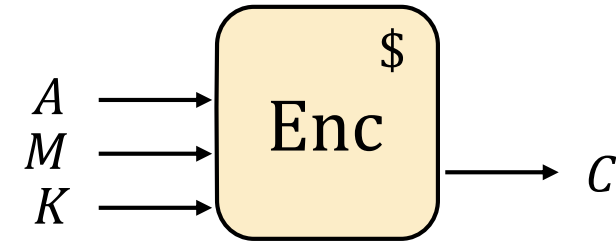
Authenticated encryption w/associated data (AEAD) – syntax

$$\text{Enc} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, M) = \text{Enc}_K(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, C) = \text{Dec}_K(C) = M/\perp$$



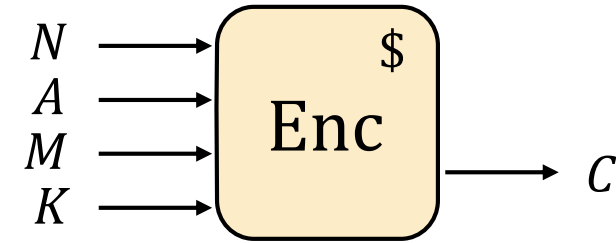
Authenticated encryption w/associated data (AEAD) – syntax

$$\text{Enc} : \mathcal{K} \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, A, M) = \text{Enc}_K(A, M) = \text{Enc}_K^A(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, A, C) = \text{Dec}_K(A, C) = \text{Dec}_K^A(C) = M/\perp$$



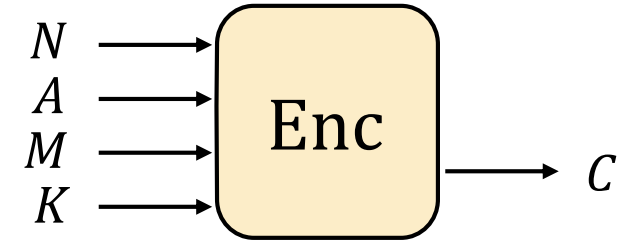
Authenticated encryption w/associated data (AEAD) – syntax

$$\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, N, A, M) = \text{Enc}_K^N(A, M) = \text{Enc}_K^{N,A}(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, N, A, C) = \text{Dec}_K^N(A, C) = \text{Dec}_K^{N,A}(C) = M/\perp$$



$\mathcal{K}$ – key space

$\mathcal{N}$ – nonce space

$\mathcal{A}$ – associated data space

$\mathcal{M}$ – message space

$\mathcal{C}$ – ciphertext space

Correctness requirement: $\forall K \in \mathcal{K}, \forall N \in \mathcal{N}, \forall A \in \mathcal{A}, \forall M \in \mathcal{M}$:

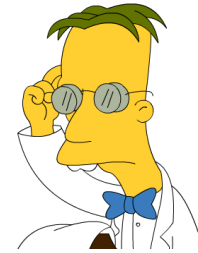
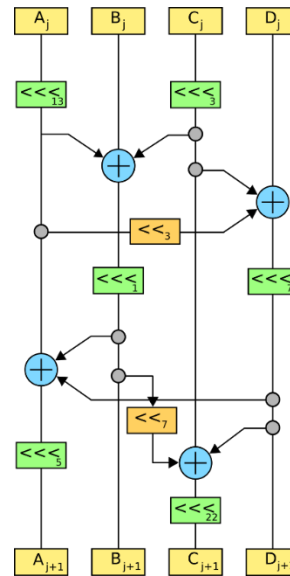
$$\text{Dec}_K^{N,A}(\text{Enc}_K^{N,A}(M)) = M$$

AEAD security (nonce-based)

$\text{Exp}_{\Sigma}^{\text{aead}}(A)$		
	$\mathcal{E}(N, A, M)$	$\mathcal{D}(N, A, C)$
1. $b \xleftarrow{\$} \{0,1\}$	1. if $N \in \text{Nonces}$ then // cheating!	1. if $(N, A, C) \in \text{Ciphertexts}$ then // cheating!
2. $\text{Nonces} \leftarrow []$	2. return $\perp$	2. return $\perp$
3. $\text{Ciphertexts} \leftarrow []$	3. $R \xleftarrow{\$} \{0,1\}^{ M }$	3. $C_0 \leftarrow \Sigma.\text{Dec}(K, N, A, C)$
4. $K \leftarrow \Sigma.\text{KeyGen}$	4. $C_0 \leftarrow \Sigma.\text{Enc}(K, N, A, M)$	4. $C_1 \leftarrow \perp$
5. $b' \leftarrow A^{\mathcal{E}(\cdot, \cdot, \cdot), \mathcal{D}(\cdot, \cdot, \cdot)}$	5. $C_1 \leftarrow \Sigma.\text{Enc}(K, N, A, R)$	5. return C_b
6. return $b' \stackrel{?}{=} b$	6. $\text{Nonces.add}(N)$	
	7. $\text{Ciphertexts.add}(N, A, C_b)$	
	8. return C_b	

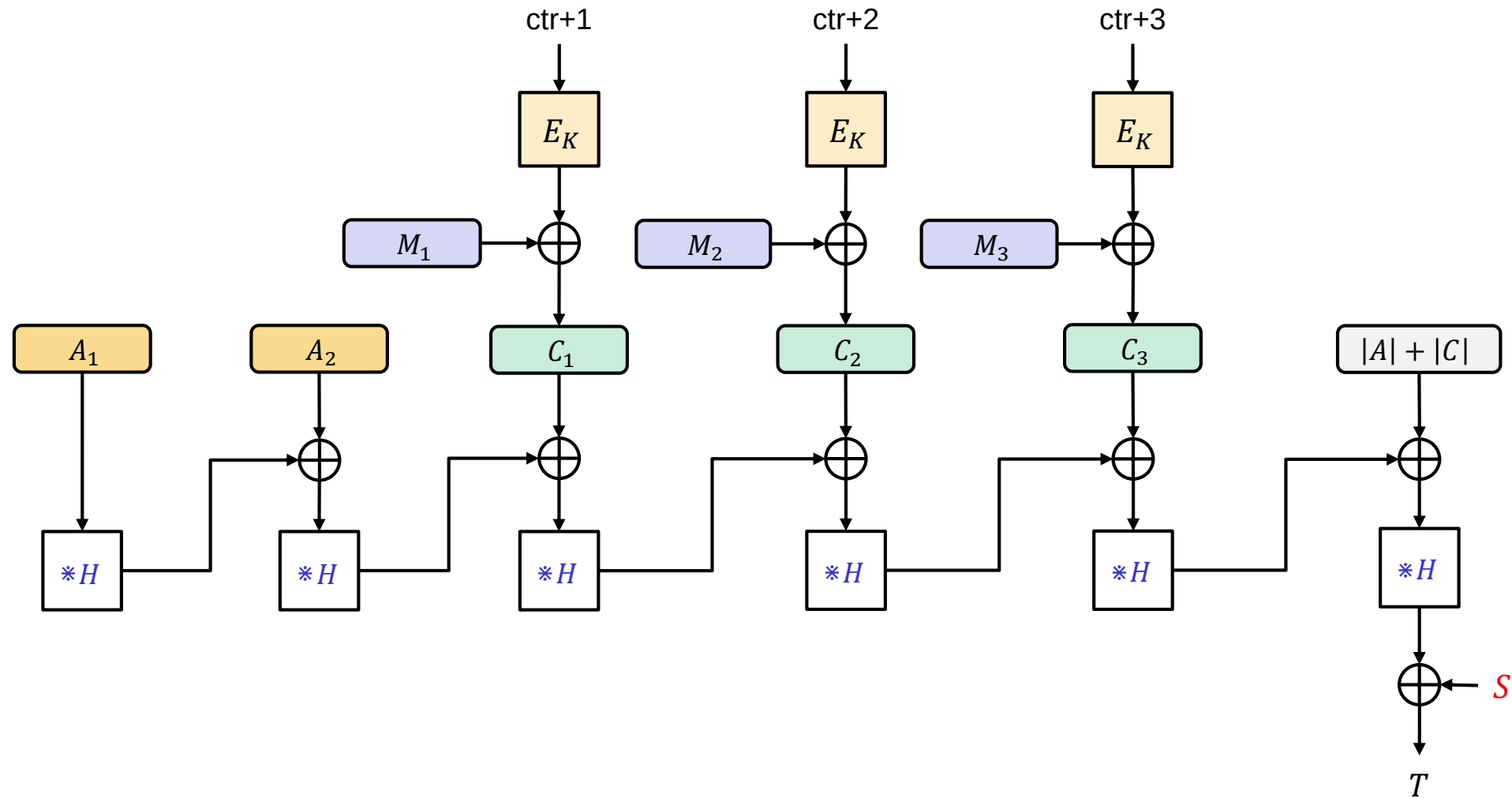
Definition: The (nonc-based) **AEAD-advantage** of an adversary A is

$$\text{Adv}_{\Sigma}^{\text{aead}}(A) = 2 \cdot \left| \Pr[\text{Exp}_{\Sigma}^{\text{aead}}(A) \Rightarrow \text{true}] - 1/2 \right|$$

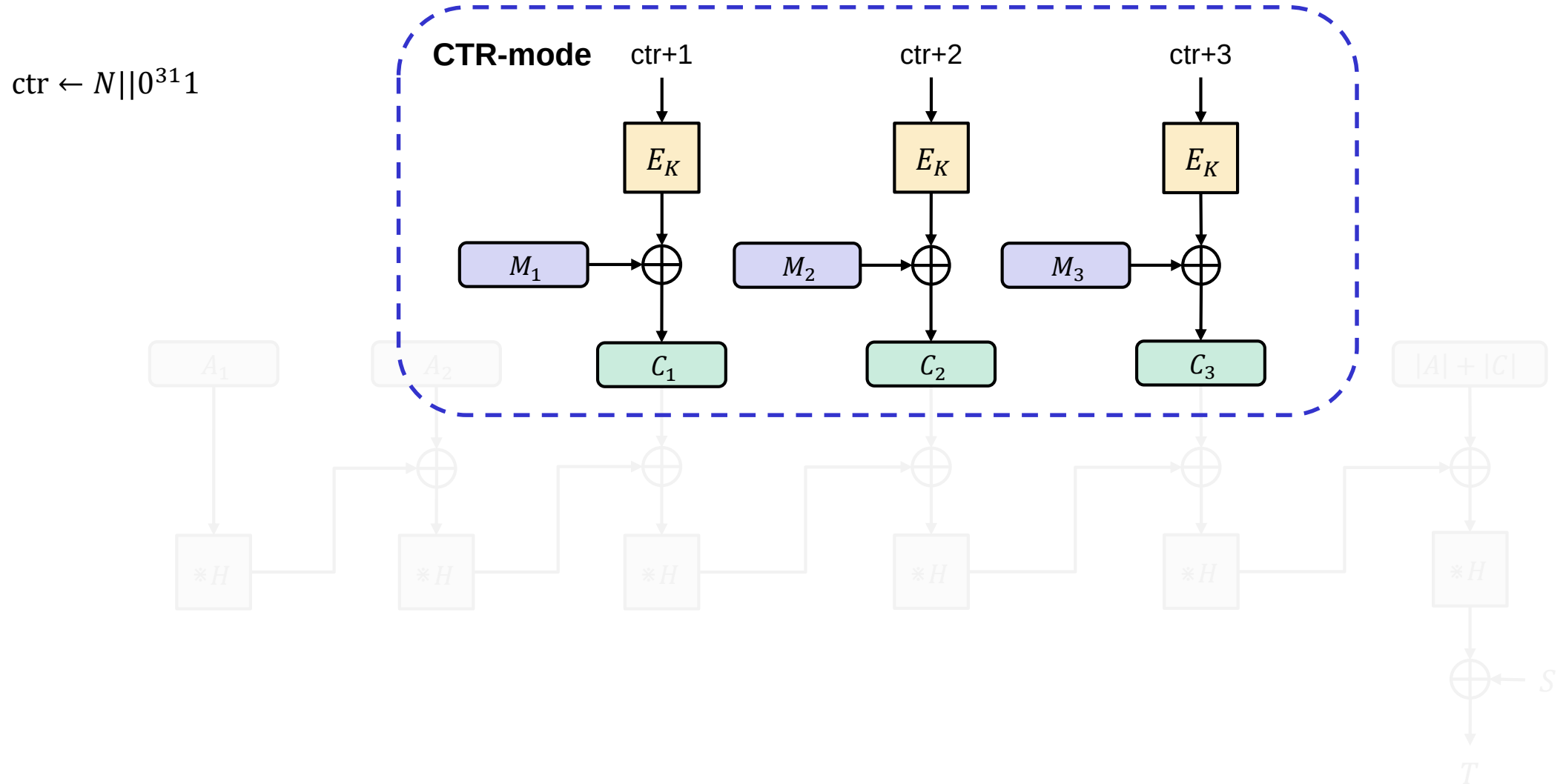


Constructing AEAD schemes

GCM – Galois/Counter Mode



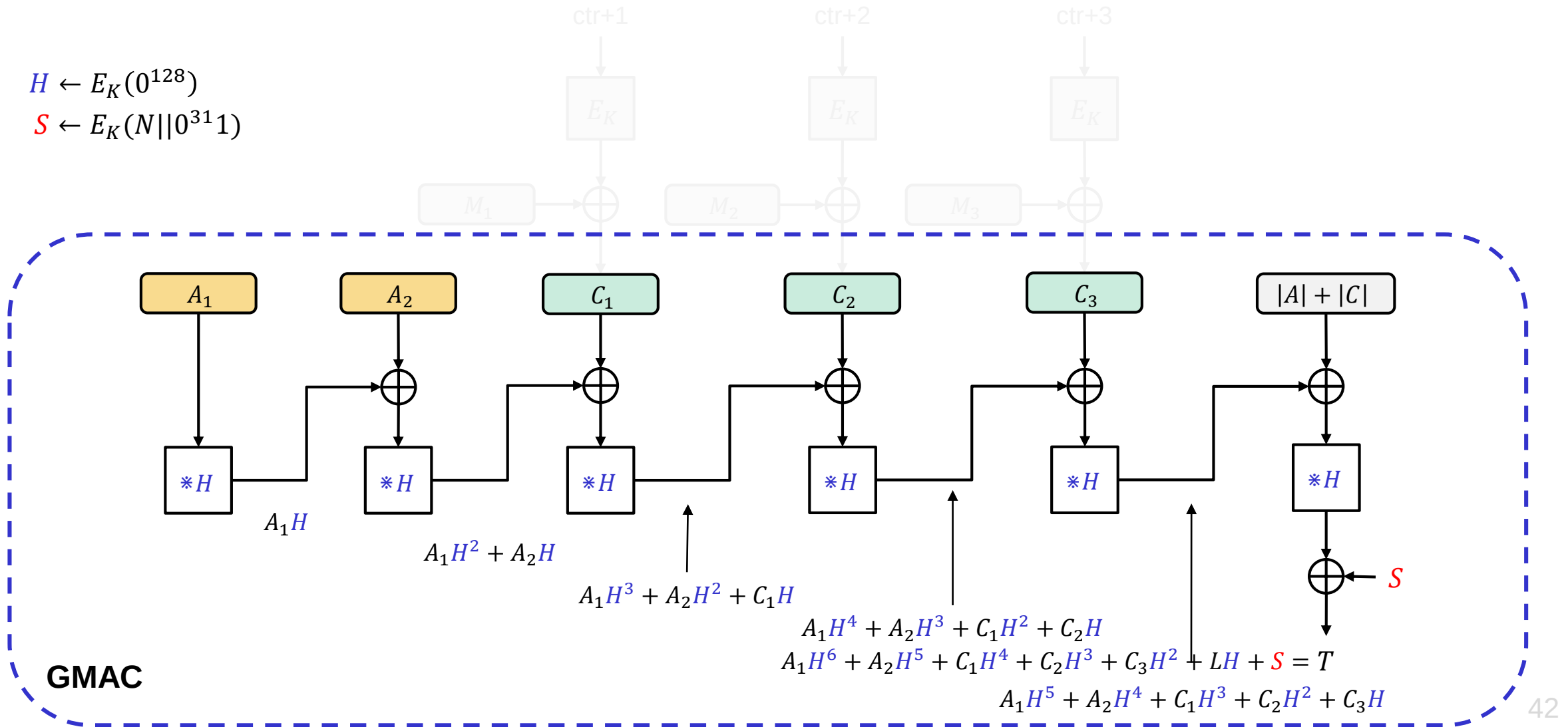
GCM – Galois/Counter Mode



GCM – Galois/Counter Mode

$$H \leftarrow E_K(0^{128})$$

$$S \leftarrow E_K(N || 0^{31}1)$$

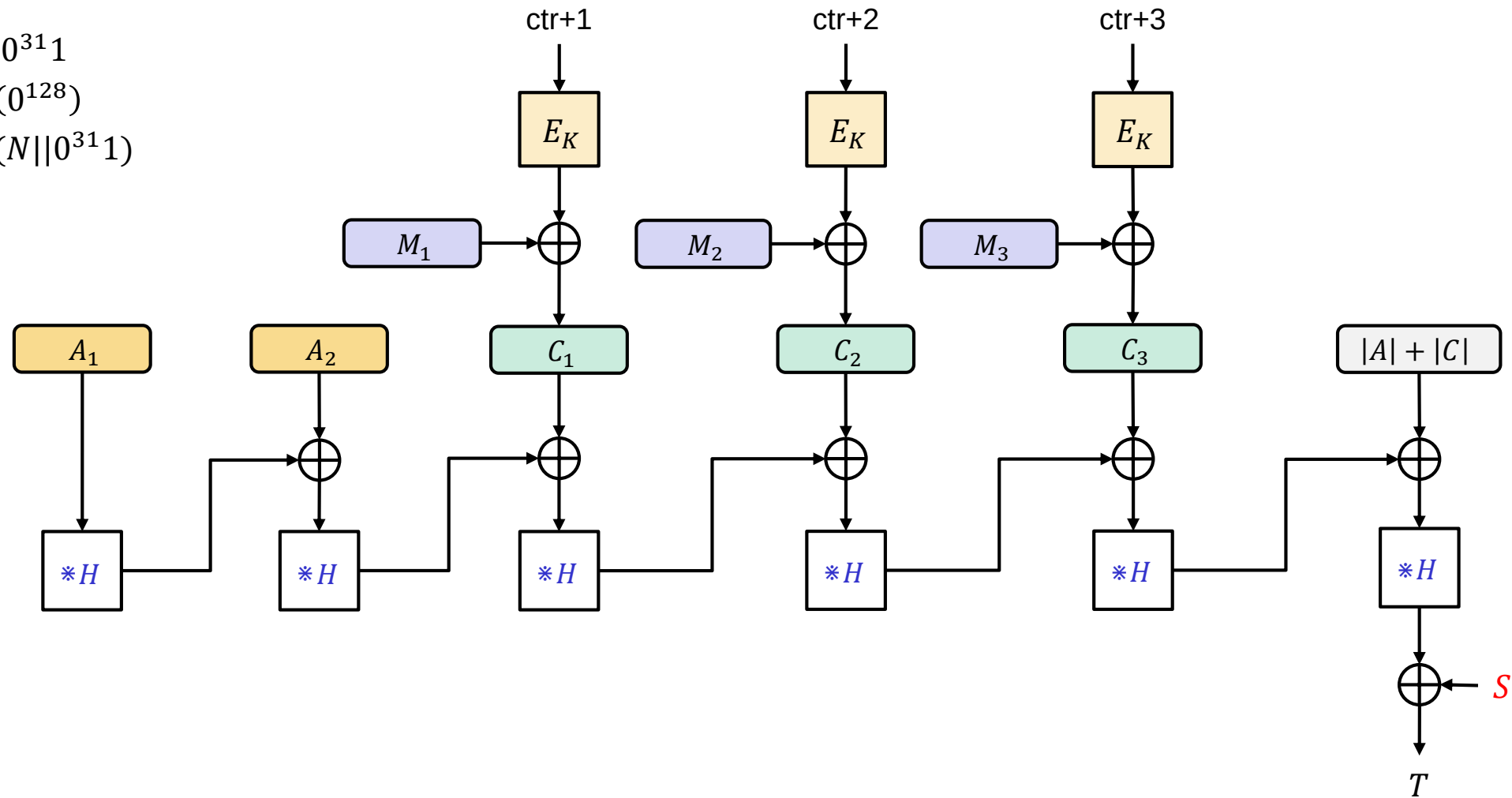


GCM – Galois/Counter Mode

$\text{ctr} \leftarrow N || 0^{31}1$

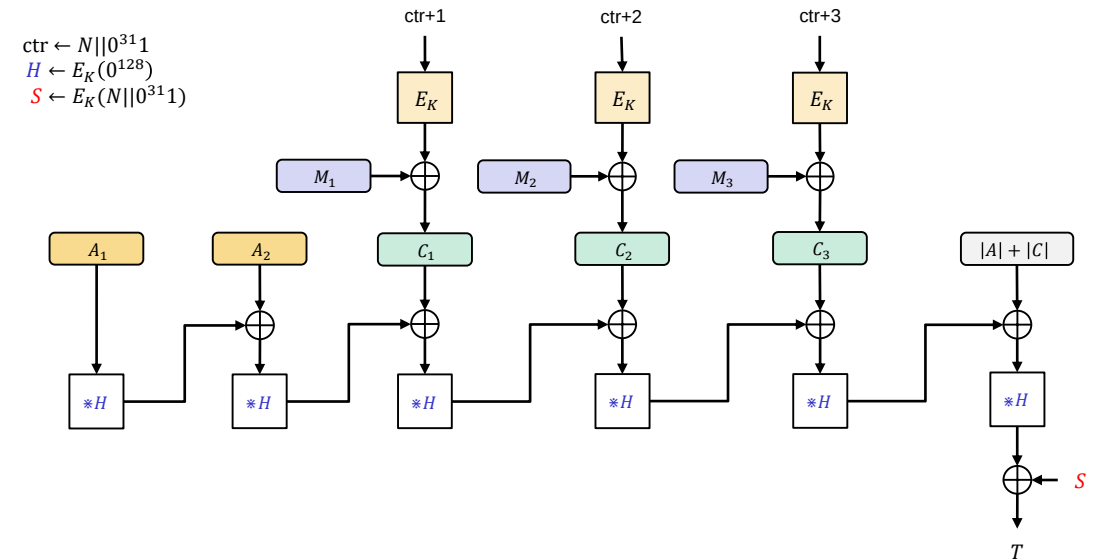
$H \leftarrow E_K(0^{128})$

$S \leftarrow E_K(N || 0^{31}1)$

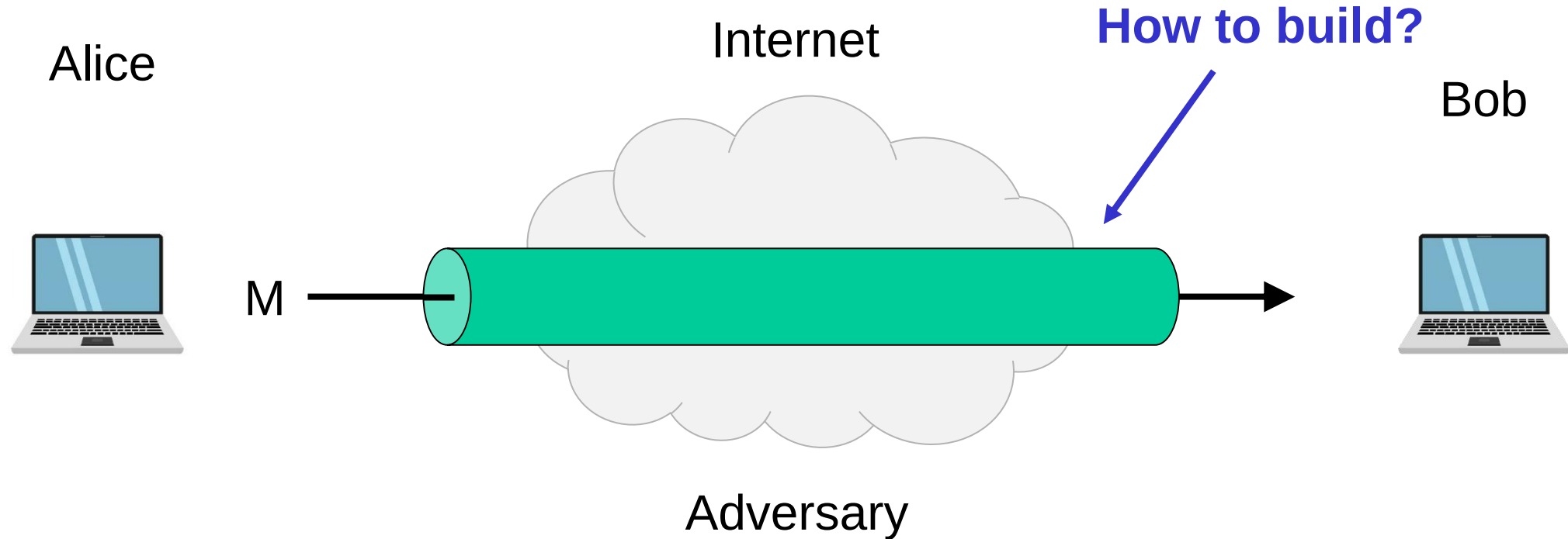


GCM – properties

- **Theorem:** AEAD-secure if E is a secure PRF
- Very fast
 - Especially with AES-NI and Intel PCLMULQDQ instructions
- Online
 - Doesn't need to know the length of the message before starting encryption
- Brittle
 - Nonce-reuse is very bad (see Problem set 5)
 - Tricky to implement correctly
- Used everywhere
 - Probably the most used mode on the Internet



Ideal solution: secure channels



Security goals:

- **Data privacy:** adversary should not be able to read message M ✓
- **Data integrity:** adversary should not be able to modify message M ✓
- **Data authenticity:** message M really originated from Alice ✓

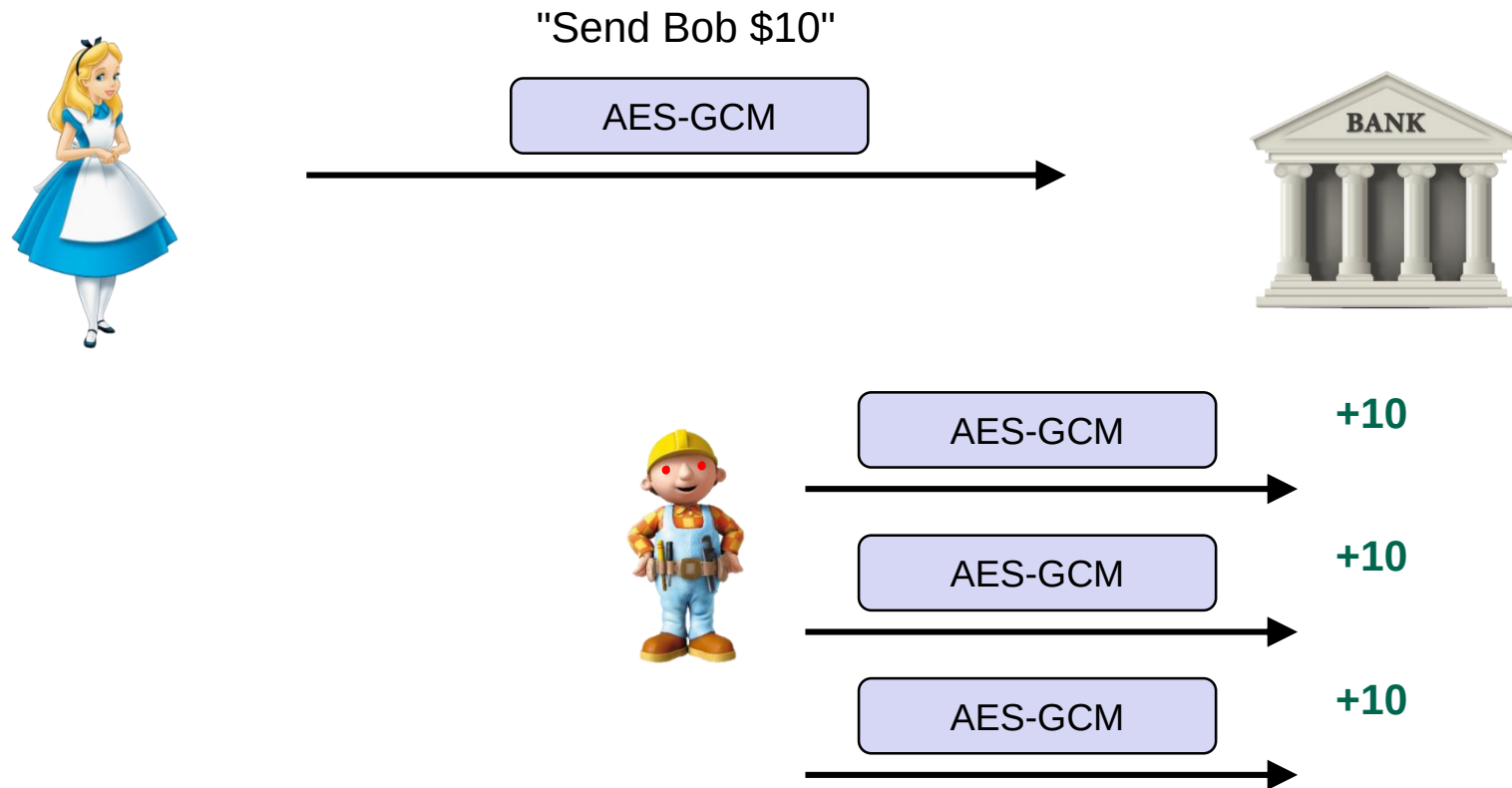
Ideal solution: secure channels



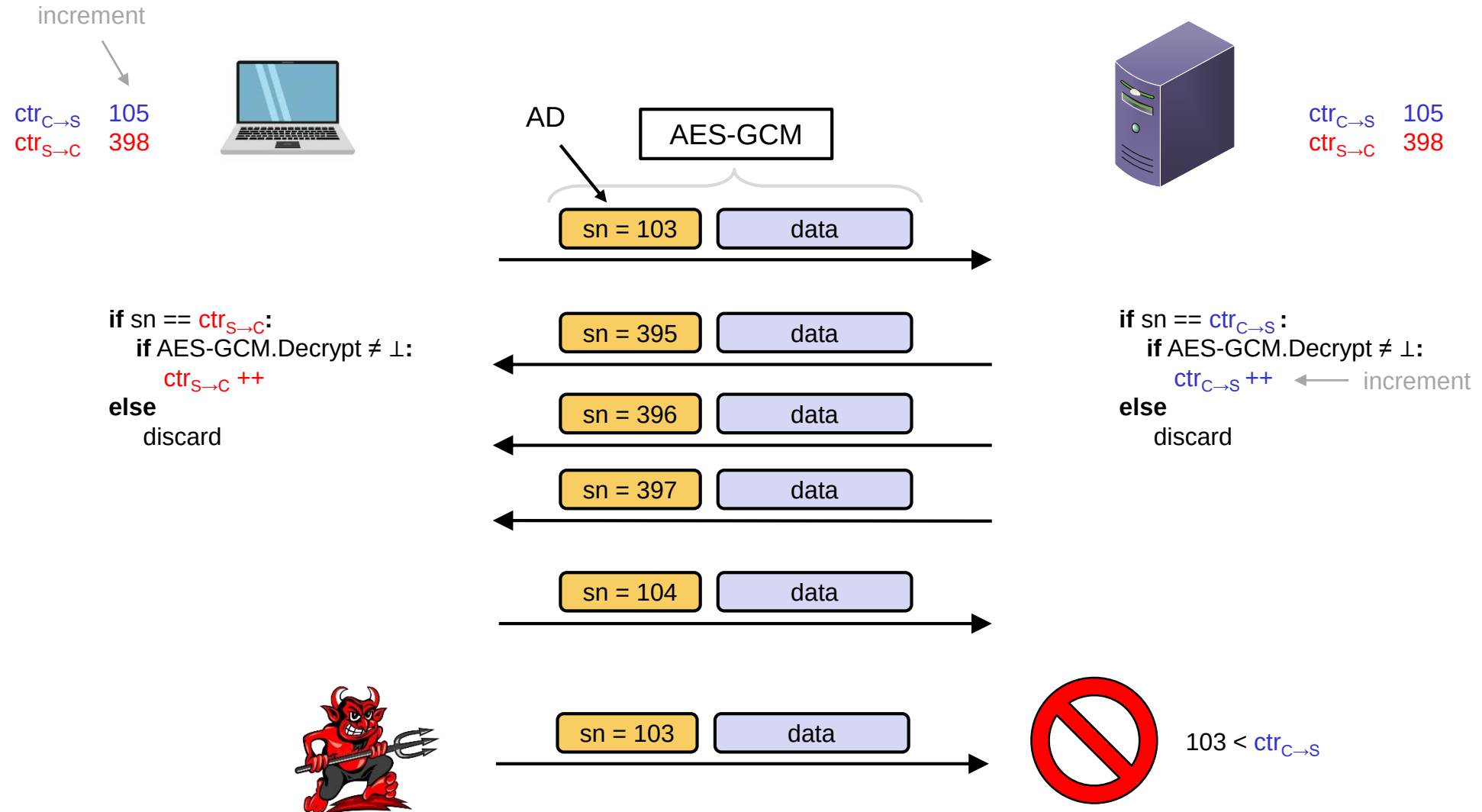
Security goals:

- **Data privacy:** adversary should not be able to read message M ✓
- **Data integrity:** adversary should not be able to modify message M ✓
- **Data authenticity:** message M really originated from Alice ✓

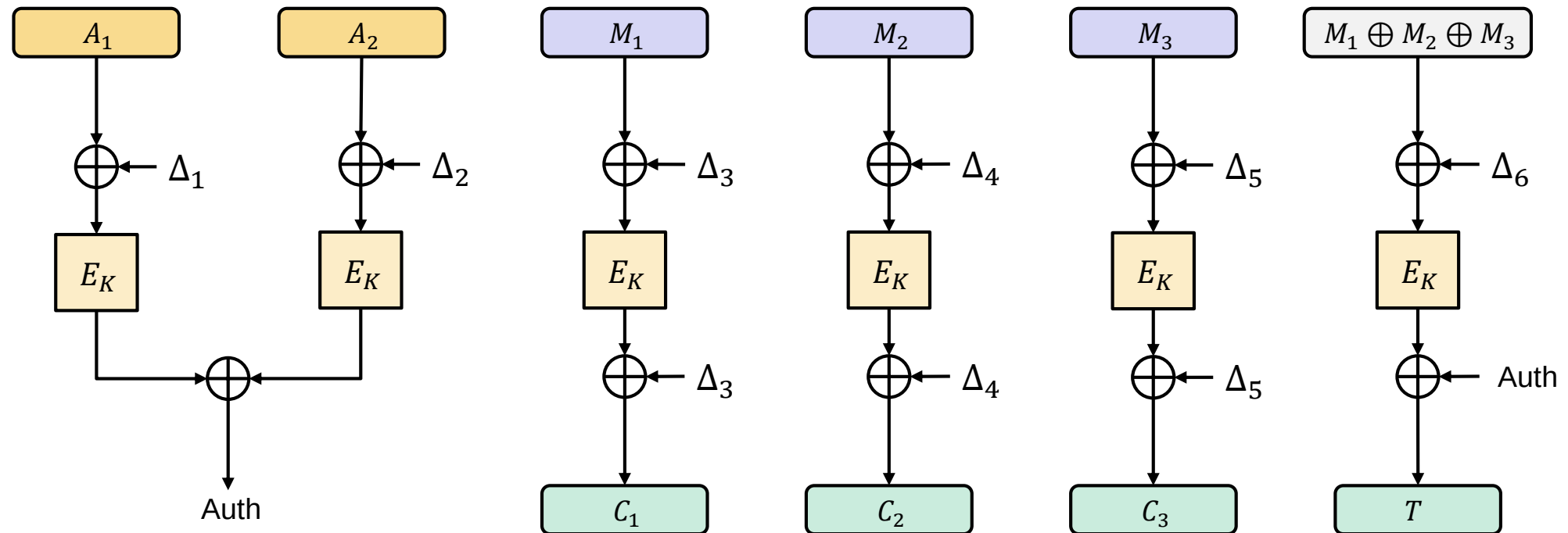
AES-GCM = secure channel?



Secure channels – TLS / IPsec



OCBv3 – Offset Codebook Mode



Δ_i - derived from 96-bit nonce N

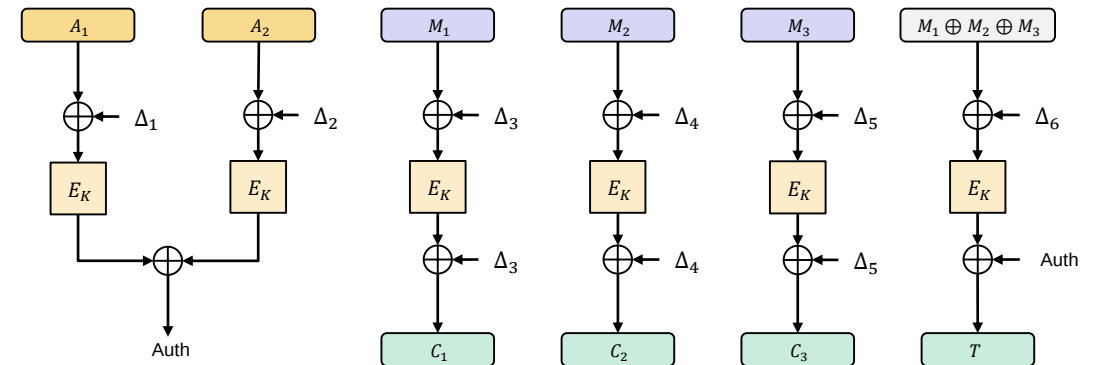
OCBv3 – properties

- **Theorem:** AE secure if E is a secure PRP
- Basically the fastest an AEAD algorithm can be
- Fully parallelizable and online
- Incremental
- Hardly used anywhere due to patents

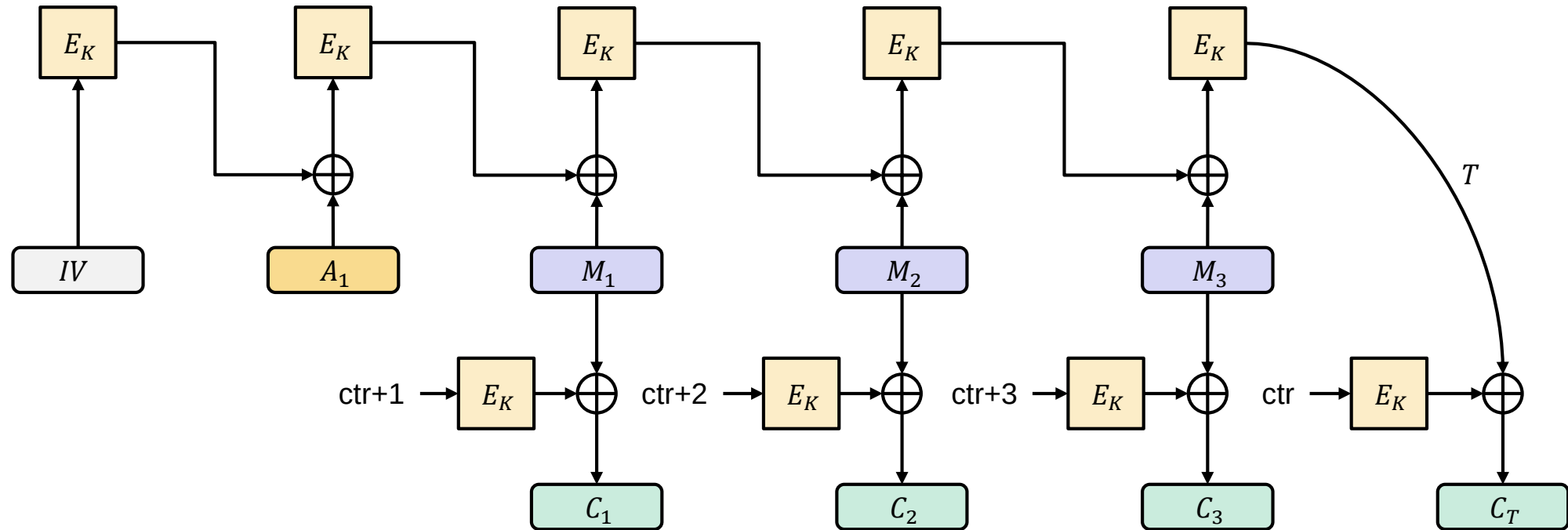
“If OCB was your kid, he’d play three sports and be on his way to Harvard. You’d brag about him to all your friends.”

“Unfortunately OCB is *not* your kid. It belongs to Philip Rogaway, who also happens to hold a patent on it.”

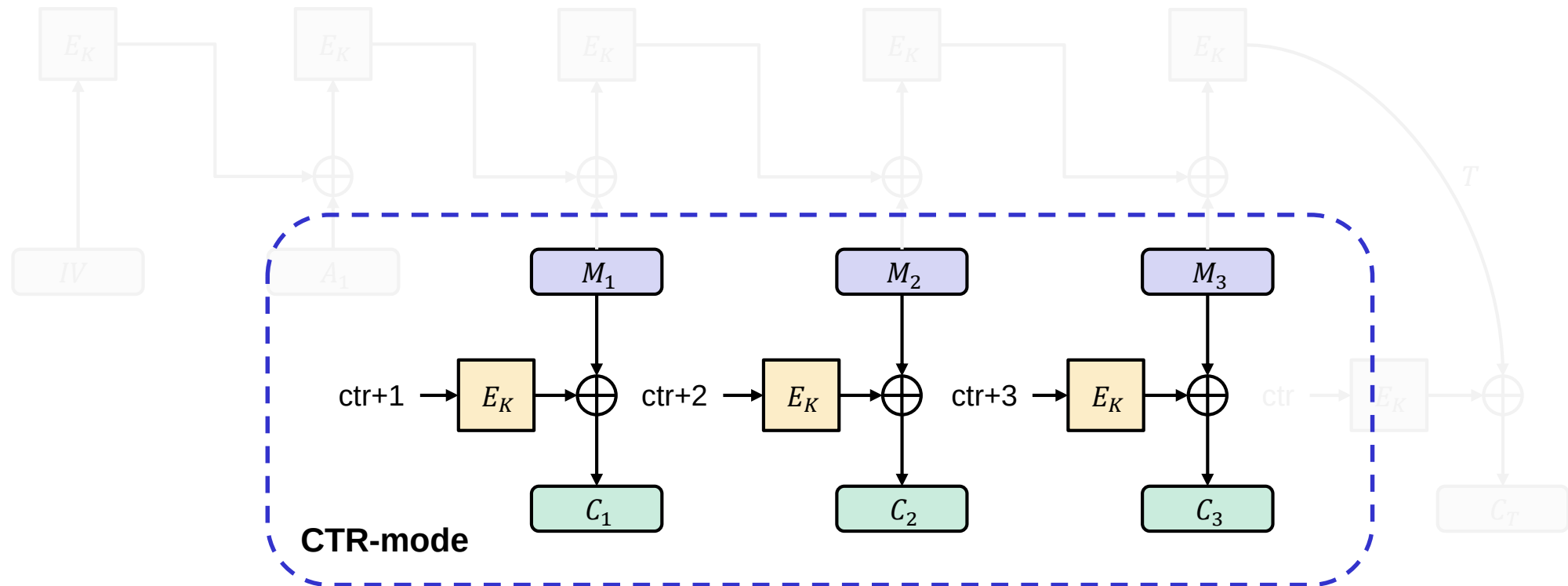
Matthew Green



CCM – Counter Mode with CBC-MAC

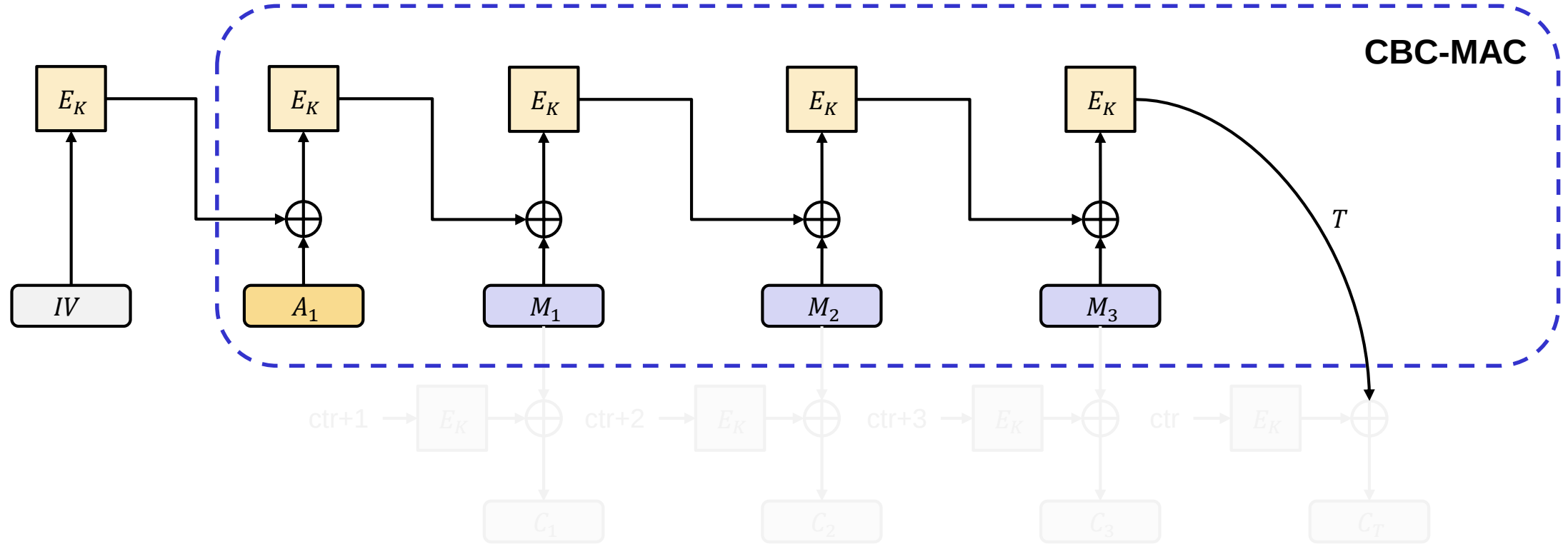


CCM – Counter Mode with CBC-MAC

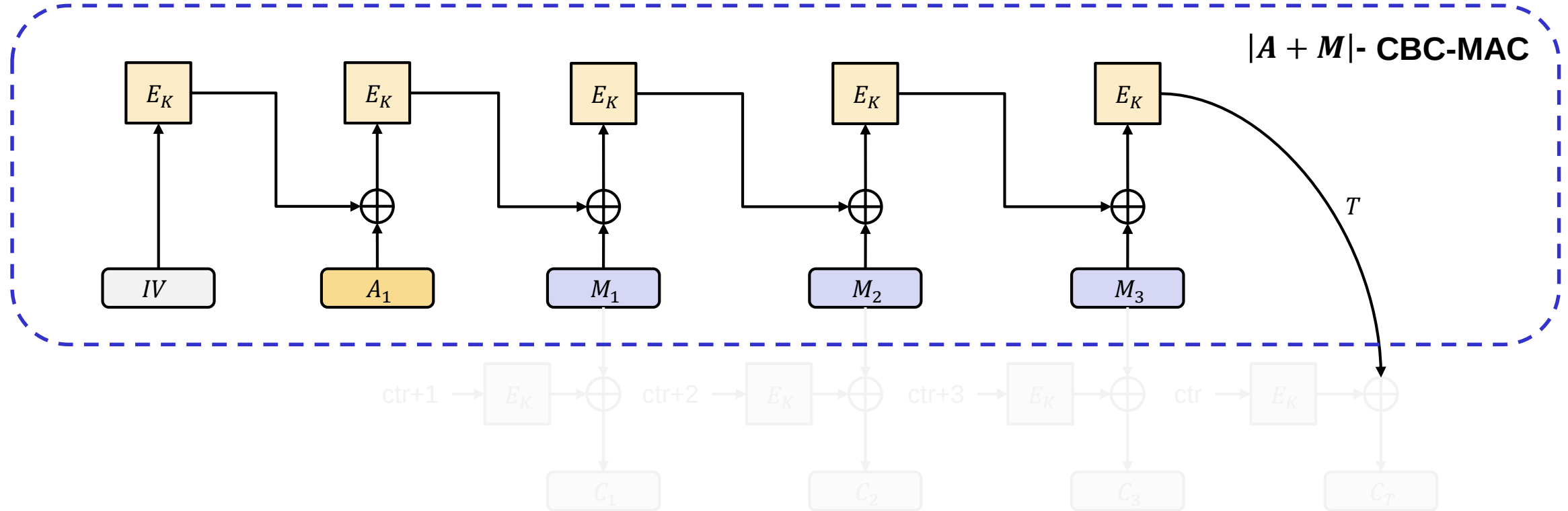


$$\text{ctr} \leftarrow F_1 || N || 0^{16}$$

CCM – Counter Mode with CBC-MAC

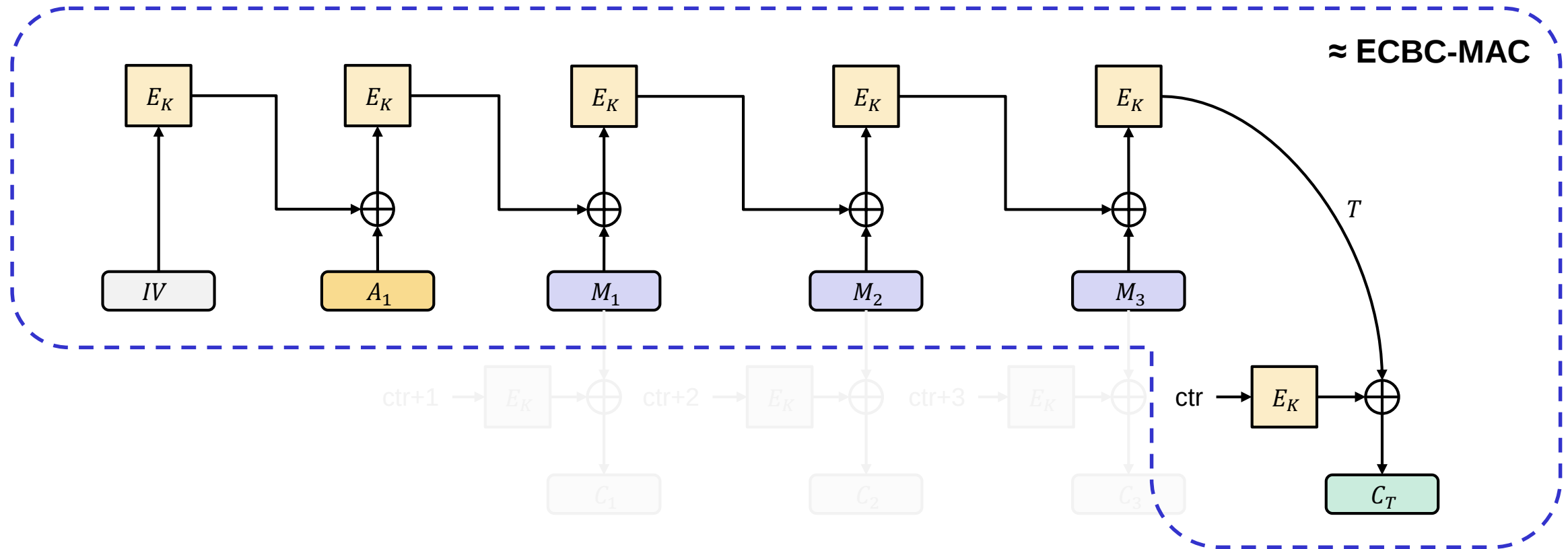


CCM – Counter Mode with CBC-MAC



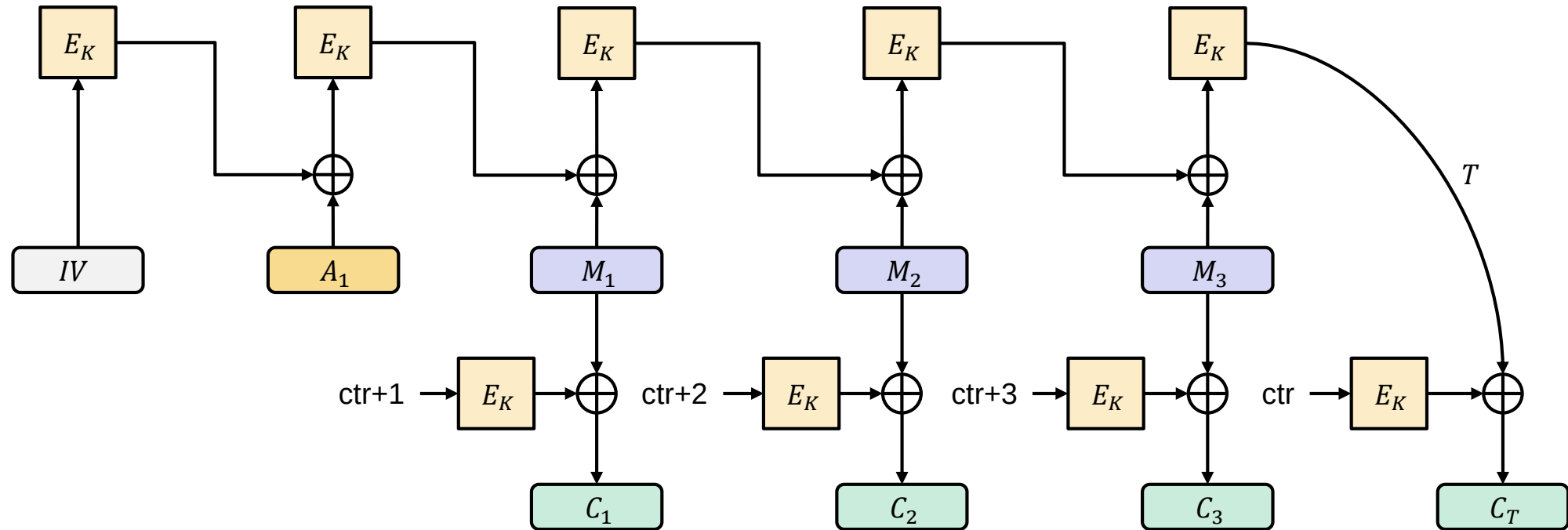
$$IV \leftarrow F_2 || N || \text{len}_{16}(AD + M)$$

CCM – Counter Mode with CBC-MAC



$$IV \leftarrow F_2 || N || \text{len}_{16}(AD + M)$$

CCM – Counter Mode with CBC-MAC



$$ctr \leftarrow F_1 || N || 0^{16}$$

$$IV \leftarrow F_2 || N || \text{len}_{16}(AD + M)$$

CCM – properties

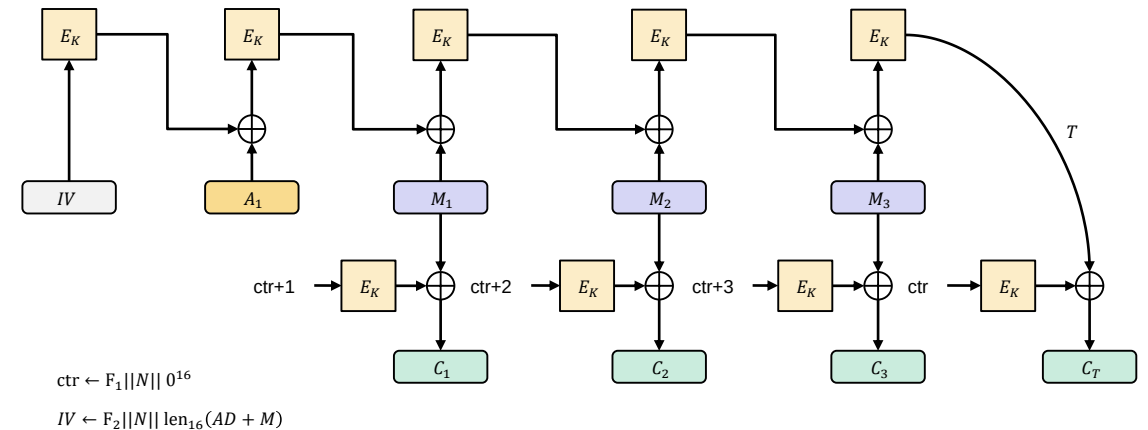
- **Theorem:** AEAD secure if E is a secure PRF

- Slow; needs 2 block cipher calls per message block

“CCM is the 1989 Volvo station wagon of AEAD modes.
It'll get you to your destination reliably, just not in a hurry.”

Matthew Green

- Sequential; not online
- Clunky message encoding
- Royalty-free
 - Designed specifically as an alternative to OCB for use in WiFi (WPA2)
- Widely used
 - Default encryption algorithm in WPA2



Summary

- Authenticated encryption: privacy + integrity in one primitive
- AEAD: AE + *associated data*: data that gets integrity protection, but not privacy protection (e.g. protocol headers)
- AEAD examples:
 - GCM
 - CCM
 - OCB
- AEAD is *not* a secure channel!
 - Does not provide *replay* protection
 - Secure channels from AEAD: add counters/nonces/timers
- Next week: hash functions